Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №5 по курсу "Анализ алгоритмов"

Тема Конвейерное программирование

Студент Козлова И.В.

Группа ИУ7-52Б

Оценка (баллы)

Преподаватель Волкова Л.Л.

Москва — 2021 г.

Содержание

Введение

Конвейер — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (увеличения числа инструкций, выполняемых в единицу времени — эксплуатация параллелизма на уровне инструкций), технология, используемая при разработке компьютеров и других цифровых электронных устройств.

Сам термин «конвейер» пришёл из промышленности, где используется подобный принцип работы — материал автоматически подтягивается по ленте конвейера к рабочему, который осуществляет с ним необходимые действия, следующий за ним рабочий выполняет свои функции над получившейся заготовкой, следующий делает ещё что-то и т.д. Таким образом, к концу конвейера цепочка рабочих полностью выполняет все поставленные задачи, сохраняя высокий темп производства. В процессорах роль рабочих исполняют функциональные модули, входящие в состав процессора.

Целью данной лабораторной работы является получение навыков организаци асинхронного взаимодействия потоков на примере ковейерной обработки данных.

Для достижения данной цели необходимо решить следующие задачи.

1. Изучения основ конвейерной обработки данных.
2. Применение изученных основ для реализации конвейерной обработки данных.
3. Получения практических навыков.
4. Получение статистики выполнения программы.
5. Выбор и обоснование языка программирования, для решения данной задачи.
6. Описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В данном разделе представлены теоретические сведения о рассматриваемых алгоритмах.

1.1 Описание конвейреной обработки данных

Конвейер [1]— способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (увеличения числа инструкций, выполняемых в единицу времени — эксплуатация параллелизма на уровне инструкций), технология, используемая при разработке компьютеров и других цифровых электронных устройств.

Идея заключается в параллельном выполнении нескольких инструкций процессора. Сложные инструкции процессора представляются в виде последовательности более простых стадий. Вместо выполнения инструкций последовательно (ожидания завершения конца одной инструкции и перехода к следующей), следующая инструкция может выполняться через несколько стадий выполнения первой инструкции. Это позволяет управляющим цепям процессора обрабатывать инструкции намного быстрее, чем при выполнении эксклюзивной полной обработки каждой инструкции от начала до конца.

1.2 Описание алгоритмов

В данной лабораторной работе предполагается создание программы на основе конвейрной обработки данных. В качестве алгоритмов на каждую из трех лент были выбраны следующие алгоритмы.

1. Поиск среднего арифмитического значения в массиве.

2. Подсчет количества числе в массиве больше, чем среднее значение.

3. Определение является ли найденное количество простым числом или нет.

1.3 Вывод

В данном разделе были рассмотрены основополагающие материалы, которые в дальнейшем потребуются для реализации алгоритмов. Были рассмотрены особенности построения конвейерных вычислений.

2 Конструкторская часть

В данном разделе будут представлены схемы рассматриваемых алгоритмов и требования к вводу.

2.1 Требования к вводу

1. На вход подаются два числа (первое - количество задач, второе - длина массива).
2. Если на вход пришли не цифры, то программа завершается.

2.2 Схема алгоритма

2.2.1 Схема конве1ерной обработки данных

Схема алгоритма главного потока представлена на рисунке 2.1.

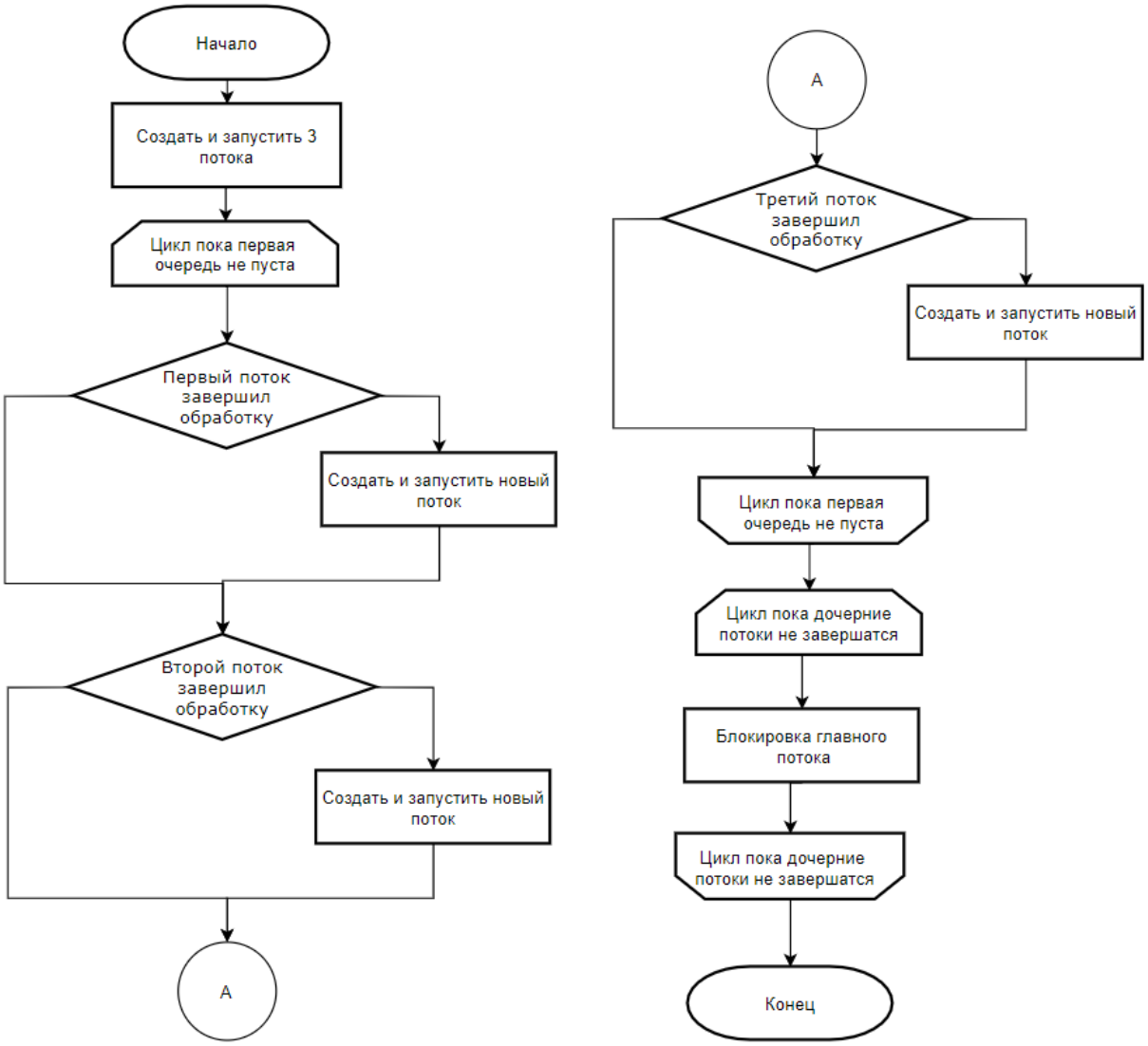

Рисунок 2.1 – Схема алгоритма главного потока

Схема алгоритма конвейера представлена на рисунке 2.2.

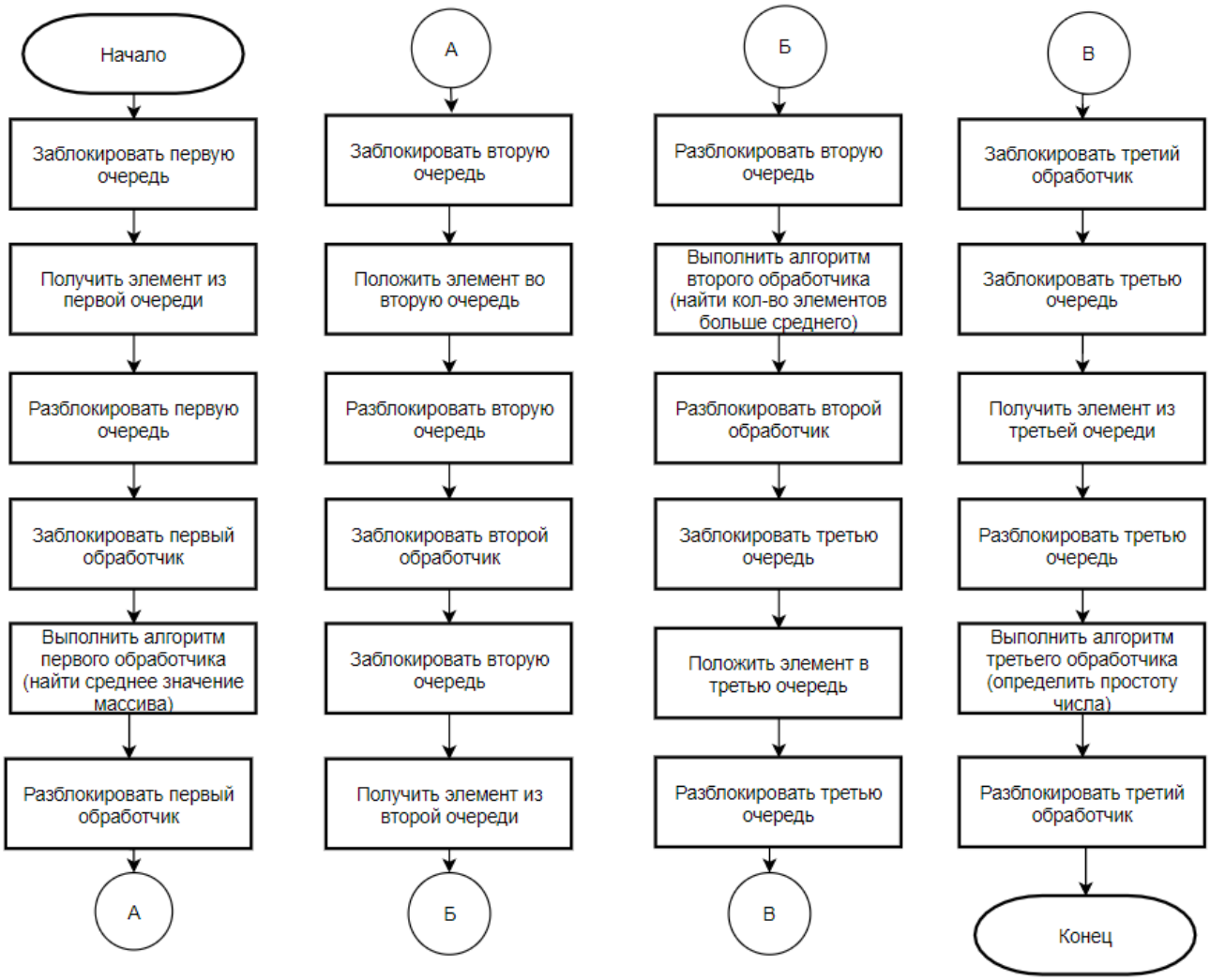

Рисунок 2.2 – Схема алгоритма конвейера

2.2.2 Схемы алгоритмов на лентах конвейера

Схема нахождения среднего арифмитического значения массива представлена на рисунке 2.3.

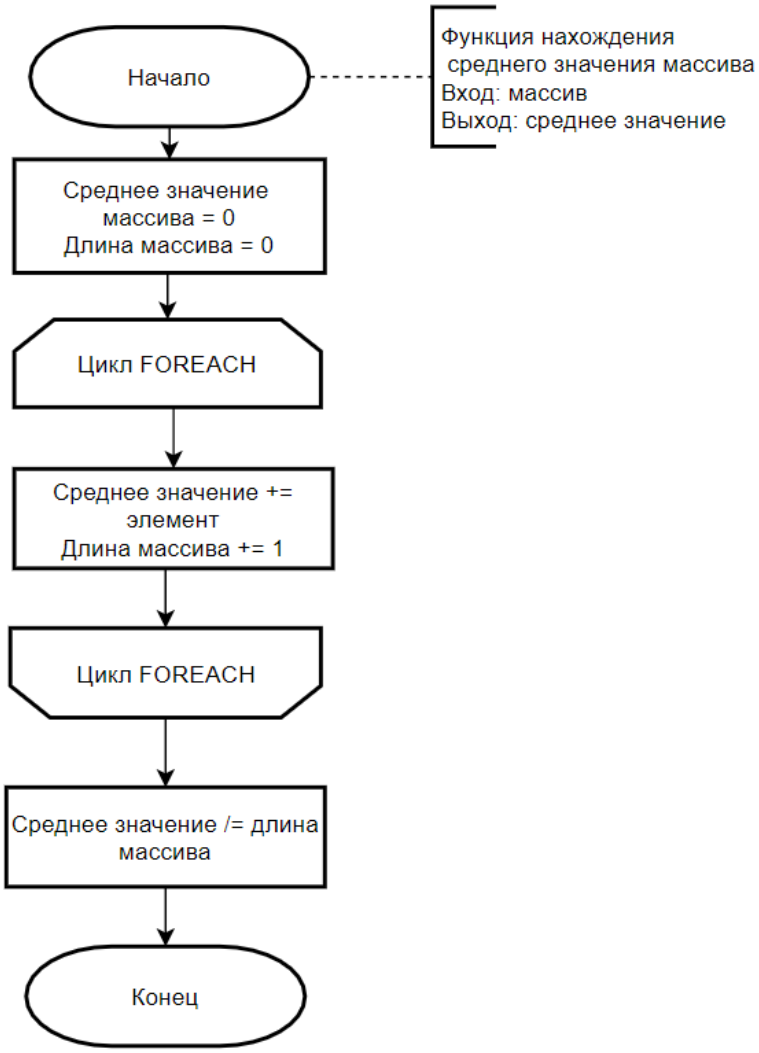

Рисунок 2.3 – Схема нахождения среднего арифмитического значения массива

Схема подсчета количества элементов в массиве, больших среднего арифметического массива, представлена на рисунке 2.4.

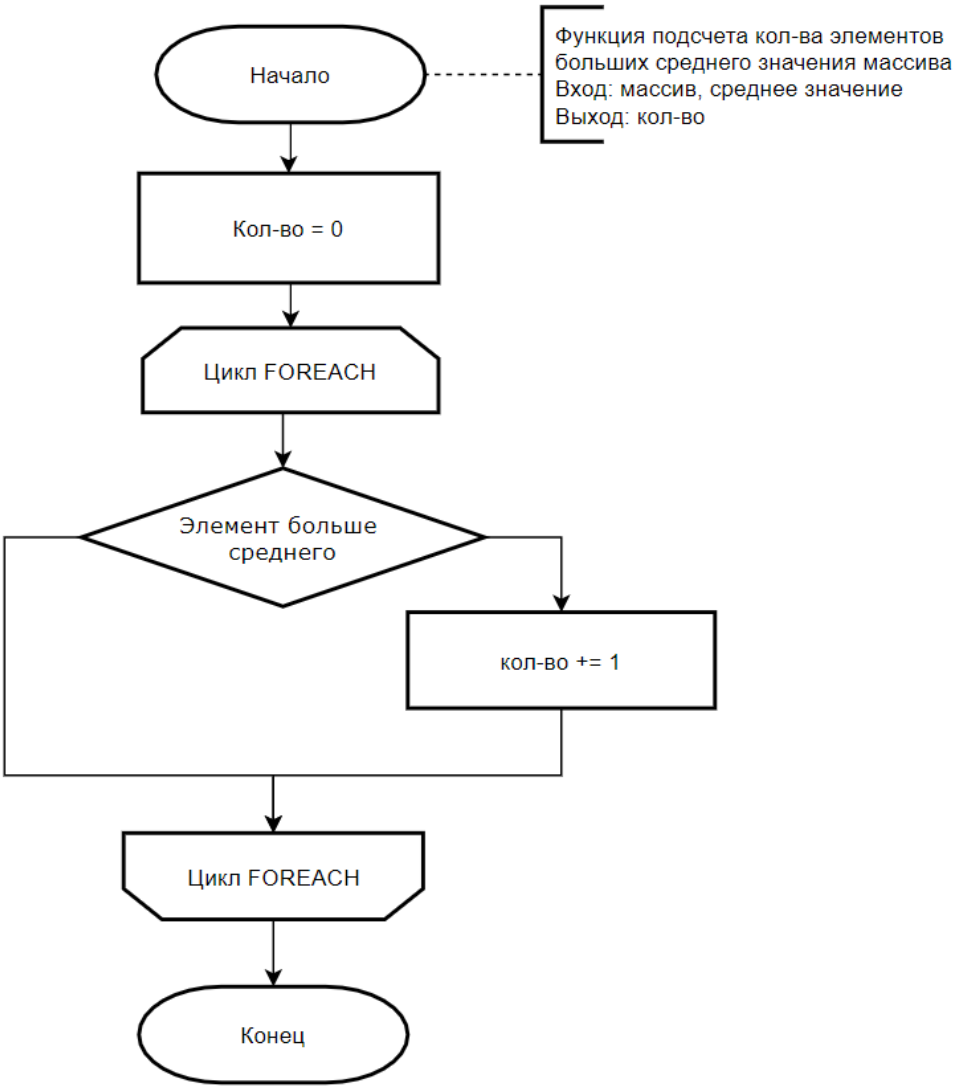

Рисунок 2.4 – Схема подсчета количества элементов в массиве, больших среднего арифметического массива

Схема определения простоты числа представлена на рисунке 2.5.

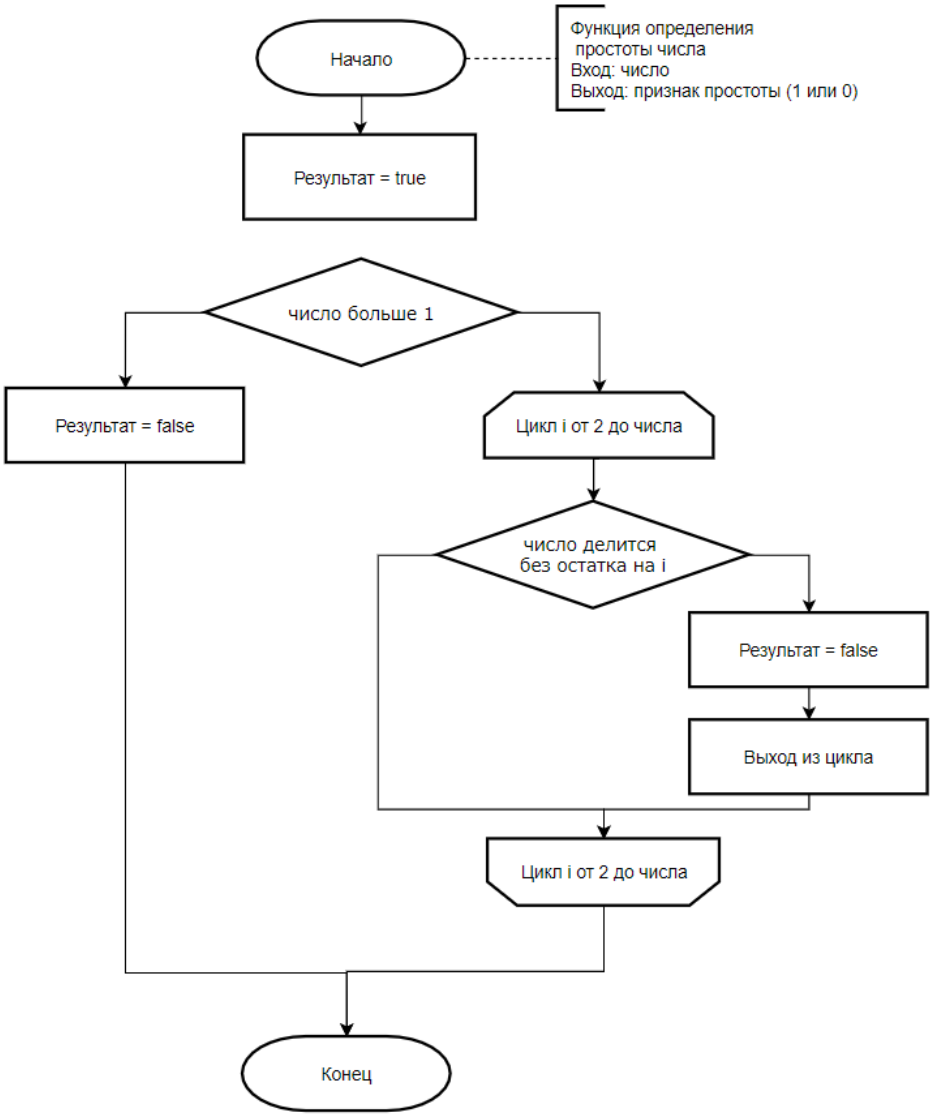

Рисунок 2.5 – Схема определения простоты числа

2.3 Вывод

В данном разделе были рассмотрены схемы однопоточной и многопоточной реализации алгоритма ранговой сортировки.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Выбор языка программирования

В данной лабораторной работе использовался язык программирования - C# [2].

Данный язык является нативным.

В качестве среды разработки выбор сделан в сторону Visual Studio 2019 [3].

Замеры времени производились с помощью структуры DateTime [4] и свойства DateTime.Now.Ticks [5].

3.2 Сведения о модулях программы

Программа состоит из одного модуля:

1. Program.cs - главный файл программы, в котором содержится точка входа в программу, генерация массивов и очередей, а также реализация конвейера.

3.3 Листинг кода

В листинге 3.1 представлена реализация конвейера.

Листинг 3.1 – Реализация конвейера

```
1 public static void Conveyor(object obj)
2 {
3     ThreadArgs args = (ThreadArgs)obj;
4     int avg = 0, count = 0, proc = 0;
5
6     intPtr array;
```

```
7
8       if (args.firstQueue.Count != 0)
9       {
10          lock (args.firstQueue)
11          {
12              array = args.firstQueue.Dequeue();
13          }
14
15          lock (FStage)
16          {
17              Int64 t1, t2;
18              t1 = DateTime.Now.Ticks;
19              Console.WriteLine("Lenta 1 \t{0} \t1 \t{1}",
20              Thread.CurrentThread.Name,
21              t1);
22
23              avg = AvgArrayInt(array);
24
25              t2 = DateTime.Now.Ticks;
26              Console.WriteLine("Lenta 1 \t{0} \t0 \t{1} \t{2}",
27              Thread.CurrentThread.Name,
28              t1, t2 - t1);
29          }
30
31          lock (args.secondQueue)
32          {
33              Int64 t = DateTime.Now.Ticks;
34              time2in.Add(t);
35              args.secondQueue.Enqueue(array);
36          }
37      }
38
39      if (args.secondQueue.Count != 0)
40      {
41          lock (SStage)
42          {
43              Int64 t1, t2;
44              t1 = DateTime.Now.Ticks;
45              Console.WriteLine("Lenta 2 \t{0} \t1 \t{1}",
46              Thread.CurrentThread.Name,
47              t1);
```

```
48
49              lock (args.secondQueue)
50              {
51                  Int64 t = DateTime.Now.Ticks;
52                  time2out.Add(t);
53                  array = args.secondQueue.Dequeue();
54              }
55
56              count = CountBigAvgInt(array, avg);
57
58              t2 = DateTime.Now.Ticks;
59              Console.WriteLine("Lenta␣2␣\t{0}␣\t0␣\t{1}␣\t{2}",
60              Thread.CurrentThread.Name,
61              t1, t2 - t1);
62          }
63
64          lock (args.thirdQueue)
65          {
66              Int64 t = DateTime.Now.Ticks;
67              time3in.Add(t);
68              args.thirdQueue.Enqueue(array);
69          }
70      }
71
72      if (args.thirdQueue.Count != 0)
73      {
74          lock (TStage)
75          {
76              Int64 t1, t2;
77              t1 = DateTime.Now.Ticks;
78              Console.WriteLine("Lenta␣3␣\t{0}␣\t1␣\t{1}",
79              Thread.CurrentThread.Name,
80              t1);
81
82              lock (args.thirdQueue)
83              {
84                  Int64 t = DateTime.Now.Ticks;
85                  time3out.Add(t);
86                  array = args.thirdQueue.Dequeue();
87              }
88
```

```
89              proc = CountIsProc(count);
90
91              t2 = DateTime.Now.Ticks;
92              Console.WriteLine("Lenta 3 \t{0} \t0 \t{1} \t{2}",
93              Thread.CurrentThread.Name,
94              t1, t2 - t1);
95          }
96      }
97  }
```

В листинге 3.2 представлен метод создания и запуска потоков, метод MainTread основного класса программы.

Листинг 3.2 – Метод создания и запуска потоков

```
1  public static void MainTread(Queue<intPtr> queue)
2  {
3      ThreadArgs args = new ThreadArgs(queue);
4
5      Thread FThread = new Thread(new
           ParameterizedThreadStart(Conveyor));
6      FThread.Name = "Potok 1";
7
8      Thread SThread = new Thread(new
           ParameterizedThreadStart(Conveyor));
9      SThread.Name = "Potok 2";
10
11     Thread TThread = new Thread(new
           ParameterizedThreadStart(Conveyor));
12     TThread.Name = "Potok 3";
13
14     FThread.Start(args);
15     SThread.Start(args);
16     TThread.Start(args);
17
18     while (args.firstQueue.Count != 0)
19     {
20         if (!FThread.IsAlive)
21         {
22             FThread = new Thread(new
                   ParameterizedThreadStart(Conveyor));
23             FThread.Name = "Potok 1";
24             FThread.Start(args);
```

```
25          }
26
27          if (!SThread.IsAlive)
28          {
29              SThread = new Thread(new
                    ParameterizedThreadStart(Conveyor));
30              SThread.Name = "Potok 2";
31              SThread.Start(args);
32          }
33
34          if (!TThread.IsAlive)
35          {
36              TThread = new Thread(new
                    ParameterizedThreadStart(Conveyor));
37              TThread.Name = "Potok 3";
38              TThread.Start(args);
39          }
40      }
41
42      FThread.Join();
43      SThread.Join();
44      TThread.Join();
45  }
```

В листинге 3.3 представлен код задачи, выполняемой на первой ленте конвейера.

Листинг 3.3 – Подсчет среднего арифметического значения в массиве

```
1  public static int AvgArrayInt(intPtr array)
2  {
3      int avg_int = 0, count = 0;
4      for (int i = 0; i < CountOfOperations; i++)
5      {
6          avg_int = 0;
7          count = 0;
8          foreach (var el in array)
9          {
10             avg_int += el;
11             count++;
12         }
13     }
14     avg_int /= count;
```

```
15      return avg_int;
16  }
```

В листинге 3.4 представлен код задачи, выполняемой на второй ленте конвейера.

Листинг 3.4 – Подсчет количества элементов в массиве больших среднего арифметического значения

```
1  public static int CountBigAvgInt(intPtr array, int num)
2  {
3      int count = 0;
4      for (int i = 0; i < CountOfOperations; i++)
5      {
6          count = 0;
7          foreach (var el in array)
8              if (el > num)
9                  count++;
10     }
11     return count;
12 }
```

В листинге 3.5 представлен код задачи, выполняемой на третьей ленте конвейера.

Листинг 3.5 – Реализация алгоритма определния числа на простоту

```
1  public static int CountIsProc(int num)
2  {
3      int res = 1;
4      for (int k = 0; k < CountOfOperations; k++)
5      {
6          if (num > 1)
7              for (int i = 2; i < num; i++)
8                  if (num % i == 0)
9                  {
10                     res = 0;
11                     break;
12                 }
13         else
14             res = 0;
15     }
16     return res;
17 }
```

3.4 Тестирование

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы сортировки. Тесты пройдены успешно.

Сокращение для таблицы: КЗ и ДМ - количество задач и длина массива, М - массив, ОР - ожидаемый результат, ДР - действительный результат.

Таблица 3.1 – Функциональные тесты

КЗ и ДМ	М	ОР	ДР
5, 4	$[1, 4, 8, 2]$	1	1
1, 3	$[9, 12, -1]$	1	1
2, 7	$[1, 3, 2, 1, 0, 1, 2]$	1	1
2, 3	$[4, 2, 1]$	0	0
1, 4	$[0, 3, -1, 3]$	1	1

В таблице 3.2 приведены тесты для функции, реализующей алгоритм на ленте 1.

Таблица 3.2 – Функциональные тесты для ленты 1

Массив	Ожидаемый результат	Действительный результат
$[1, 4, 8, 2]$	3	3
$[9, 12, -1]$	6	6
$[1, 3, 2, 1, 0, 1, 2]$	1	1
$[4, 2, 1]$	2	2
$[0, 3, -1, 3]$	1	1

В таблице 3.3 приведены тесты для функции, реализующей алгоритм на ленте 2.

Таблица 3.3 – Функциональные тесты для ленты 2

Массив и число	Ожидаемый результат	Действительный результат
$[1, 4, 8, 2]$, 3	2	2
$[9, 12, -1]$, 6	2	2
$[1, 3, 2, 1, 0, 1, 2]$, 1	3	3
$[4, 2, 1]$, 2	1	1
$[0, 3, -1, 3]$, 1	3	3

В таблице 3.4 приведены тесты для функции, реализующей алгоритм на ленте 3.

Таблица 3.4 – Функциональные тесты для ленты 3

Число	Ожидаемый результат	Действительный результат
2	1	1
2	1	1
3	1	1
1	0	0
3	1	1

3.5 Вывод

В данном разделе были разобраны листинги показывающие работу конвейера, а также каждой ленты.

4 Исследовательская часть

В данном разделе будет произведено сравнение вышеизложенного алгоритма (однопоточная и многопоточная реализация).

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование, следующие.

- Операционная система: Windows 10 [6] x86_64.
- Память: 8 GiB.
- Процессор: 11th Gen Intel® Core™ i5-1135G7 @ 2.40GHz [7].
- 4 физических ядра и 8 логических ядра.

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Пример выполнения программы

На рисунке 4.1 представлен результат работы программы.

```
Input count of arrays:
3
Input len of arrays:
10

Массивы по задачам:
1: 904, 854, 456, 476, 553, 249, 512, 495, 639, 617,
2: 646, 492, 927, 216, 375, 795, 935, 45, 961, 934,
3: 977, 445, 297, 256, 739, 665, 806, 996, 103, 415,

Оператор 1      637703235751285768      5853
Оператор 2      637703235751295163      6002
Оператор 3      637703235751302610      1833
Оператор 1      637703235751305871      3589
Оператор 2      637703235751311452      3224
Оператор 3      637703235751316293      428
Оператор 1      637703235751317943      4925
Оператор 2      637703235751324610      4600
Оператор 3      637703235751329936      618
Простая реализация: 46066

Process:
Лента 1         Поток 2         1       637703235751695260
Лента 1         Поток 2         0       637703235751695260      18891
Результат: 575
Лента 1         Поток 1         1       637703235751727160
Лента 2         Поток 2         1       637703235751727275
Лента 1         Поток 1         0       637703235751727160      14381
Результат: 632
Лента 2         Поток 2         0       637703235751727275      25589
Результат: 4
Лента 3         Поток 2         1       637703235751796828
Лента 1         Поток 3         1       637703235751777388
Лента 2         Поток 1         1       637703235751798182
Лента 3         Поток 2         0       637703235751796828      8183
Результат: 0
Лента 2         Поток 1         0       637703235751798182      30648
Результат: 6
Лента 1         Поток 3         0       637703235751777388      43573
Результат: 569
Лента 2         Поток 3         1       637703235751907652
Лента 3         Поток 1         1       637703235751897826
Лента 2         Поток 3         0       637703235751907652      12046
Результат: 5
Лента 3         Поток 1         0       637703235751897826      37717
Результат: 0
Лента 3         Поток 3         1       637703235752023504
Лента 3         Поток 3         0       637703235752023504      21639
Результат: 1
Конвейер: 754384
```

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения алгоритмов

Для проведения временного анализа, программа запускалась для 10 задач.

Результаты сравнения времени работы алгоритмов последовательного и конвейерного, приведены в таблице 4.1.

Таблица 4.1 – Время (такт) выполнения параллельного и последовательного конвейеров в зависимости от длины очереди

	Тип алгоритма	
Размер	**Последовательный**	**Конвейерный**
10	34783	566716
25	64132	589083
50	177235	908572
500	733111	1444050
10000	23997915	23651996
50000	138152405	137199030
100000	238140692	211956092
500000	1368964823	1195390050

На рисунках 4.2 и 4.3 представлены графики зависимости времени от размера массива для последовательного и конвейерного алгоритма.

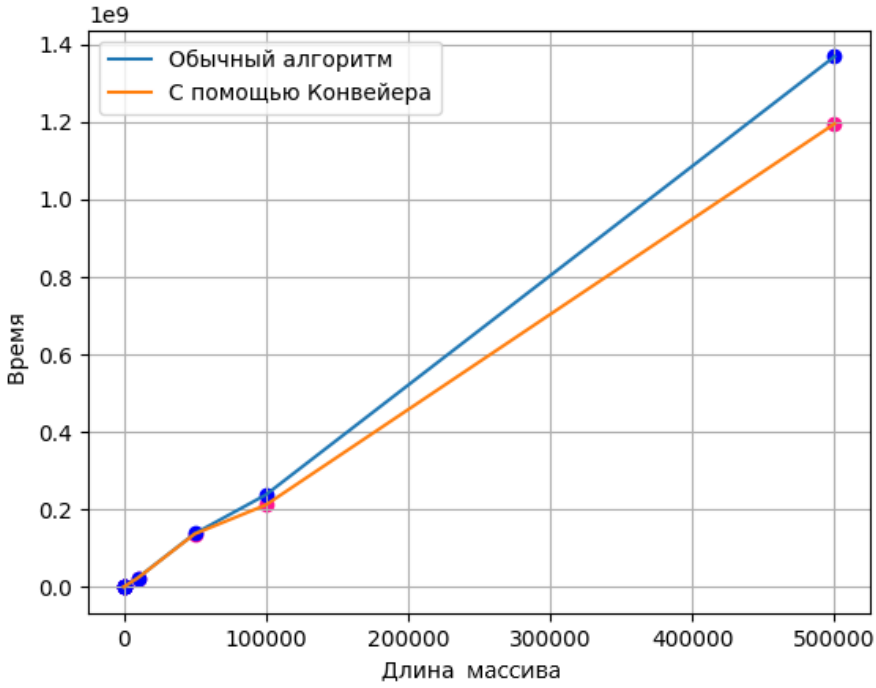

Рисунок 4.2 – График зависимости времени (такт) от размера массива

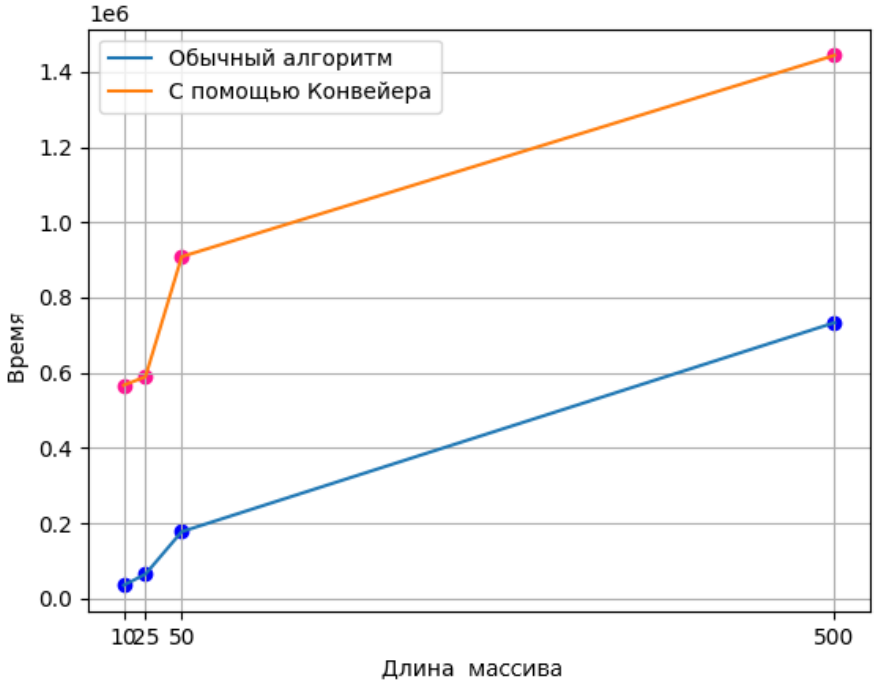

Рисунок 4.3 – График зависимости времени (такт) от размера массива

В таблице 4.2 представлены времена нахождения в очереди на определенную ленту конвейера.

Таблица 4.2 – Время (такт) нахождения в очереди на определенную ленту конвейера

Номер задачи	**Номер ленты**		
	1	**2**	**3**
1	595522	2791	1981
2	570518	9584	2223
3	570492	12343	2121
4	673939	10318	2288
5	739459	10189	8033
6	814446	11295	630
7	1099212	17404	509
8	1401780	24310	12029
9	1702263	296978	3466
10	2012547	13555	2028

Как можно заметить, время нахождения в очереди на ленте 1 самое большое, потому что все задачи сначала попадают в очередь ленты 1, и начинаются обрабатываться не сразу.

В таблице 4.3 представлены времена обработки задачи на каждой ленте конвейера.

Таблица 4.3 – Время (такт) обработки задачи на каждой ленте конвейера

Номер задачи	**Номер ленты**		
	1	**2**	**3**
1	18097	28801	298
2	43810	61235	152
3	106901	62014	155
4	71789	71040	8127
5	81812	276643	165
6	305569	285461	227
7	303807	306326	6174
8	301562	286143	654
9	29343	323545	169
10	333741	38007	244

Как можно заметить, лента 3 обработывает задачи быстрее первых двух лент. Объяснить это можно тем фактом, что трудоемкость алгоритма третьей ленты зависит от числа, поступающего на вход данного алгоритма.

Вывод

В данном разделе было произведено сравнение последовательной реализации трех алгоритмов и конвейера с использованием многопоточности. По результатам исследования можно сказать, что конвейерную обработку выгоднее применять на больших числах (большие длины массивов, большое количество задач), так как на малых размерах последовательный алгоритм выигрывает у конвейерного.

Заключение

В данной лабораторной работе были рассмотрены основополагающие материалы которые в дальнейшем потребовались при реализации конвейера.

В ходе выполнения лабораторной работы были решены следующие задачи:

1. Изучены основы конвейерной обработки данных.
2. Применены изученные основы основы для реализации конвейерной обработки данных.
3. Получены практические навыки.
4. Произведен сравнительный анализ простой и конвейерной реализации данных алгоритмов.
5. Экспериментально подтверждены различия во временной эффективности реализации простого и конвейерного алгоритма выполнения алгоритмов.
6. Подготовлен отчет о лабораторной работе.

Поставленная цель достигнута.

Литература

[1] Конвейерная организация [Электронный ресурс]. Режим доступа: http://www.citforum.mstu.edu.ru/hardware/svk/glava_5.shtml (дата обращения: 18.10.2021).

[2] Документация по C# [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения: 30.09.2021).

[3] Visual Studio Microsoft [Электронный ресурс]. Режим доступа: https://visualstudio.microsoft.com/ru/downloads/ (дата обращения: 30.09.2021).

[4] СТруктура DateTime [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/api/system.datetime?view=netframework-4.8 (дата обращения: 09.10.2021).

[5] Свойство DateTime.Now.Ticks [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/api/system.datetime.ticks?view=netcore-3.1 (дата обращения: 09.10.2021).

[6] Windows [Электронный ресурс]. Режим доступа: https://www.microsoft.com/ru-ru/windows (дата обращения: 30.09.2021).

[7] Процессор Intel® Core™ i5-1135G7 [Электронный ресурс]. Режим доступа: https://www.intel.ru/content/www/ru/ru/products/sku/208658/intel-core-i51135g7-processor-8m-cache-up-to-4-20-ghz/specifications.html (дата обращения: 04.09.2021).