

**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»**

## **Лабораторная работа № 5 по дисциплине «Анализ алгоритмов»**

**Тема Конвейерная обработка**

**Студент Калашников С.Д.**

**Группа ИУ7-53Б**

**Преподаватель Волкова Л.Л., Строганов Ю.В.**

Москва, 2022

## СОДЕРЖАНИЕ

<b>Введение</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>4</b>
1.1 Конвейерная обработка данных	4
1.2 Последовательная обработка данных	4
1.3 Описание используемых алгоритмов	4
1.4 Организация взаимодействия параллельных потоков	5
<b>2 Конструкторская часть</b>	<b>6</b>
2.1 Описание используемых структур	6
2.2 Описание алгоритмов	6
<b>3 Технологическая часть</b>	<b>13</b>
3.1 Требования к программному обеспечению	13
3.2 Средства реализации	13
3.3 Сведения о модулях программы	14
3.4 Реализация алгоритмов	14
3.5 Функциональное тестирование	19
<b>4 Исследовательская часть</b>	<b>20</b>
4.1 Технические характеристики	20
4.2 Демонстрация работы программы	20
4.3 Время выполнения реализаций алгоритмов	21
4.4 Вывод	22
<b>Заключение</b>	<b>23</b>
<b>Список использованных источников</b>	<b>24</b>

# Введение

Использование параллельной обработки уменьшает время выполнения программы. Конвейерная обработка является одним из примеров, где использование параллелизации помогает ускорить обработку данных. Суть та же, что и при работе реальных конвейерных лент, — материал поступает на обработку, после окончания обработки материал передается следующему обработчику, при этом предыдущий обработчик не ждёт полного цикла обработки материала, а получает новый материал и работает с ним.

Целью данной лабораторной работы является получение навыка организации асинхронного обмена данными между нативными потоками.

Для достижения поставленной цели требуется решить ряд задач:

- 1) описать основные принципы конвейерной обработки;
- 2) описать последовательный алгоритм обработки данных;
- 3) разработать обработчики заявок в составе конвейера и алгоритмы, выполняемые на них;
- 4) реализовать заданные алгоритмы и конвейер;
- 5) провести замеры времени для последовательного алгоритма обработки данных и для алгоритма конвейерной обработки данных;
- 6) выполнить анализ полученных результатов;
- 7) по итогам работы составить отчет.

# **1 Аналитическая часть**

В данном разделе будут рассмотрены главные принципы конвейерной обработки и параллельного доступа к данным.

## **1.1 Конвейерная обработка данных**

Конвейеризация (или конвейерная обработка) — это такая организация выполнения операций над объектами, при которой весь процесс воздействия разделяется на последовательность стадий с целью повышения производительности путём одновременного независимого выполнения операций над несколькими объектами, проходящими различные стадии. Конвейером также называют средство продвижения объектов между стадиями при такой организации.

## **1.2 Последовательная обработка данных**

При последовательной обработке данных заявки попадают в очереди последовательно друг за другом. Пока одна заявка не будет обработана, другие заявки не смогут попасть в какие-либо очереди и вынуждены ждать.

## **1.3 Описание используемых алгоритмов**

В качестве примера для операции, подвергающейся конвейерной обработке, будет обрабатываться строка, разделенная на символы.

В программе будут использованы следующие алгоритмы:

- шифровка входной строки шифром Цезаря;
- шифровка входной строки *xor* шифром;
- шифровка входной строки шифром Цезаря.

Шифр Цезаря — это вид шифра подстановки, в котором каждый символ в открытом тексте заменяется символом, находящимся на некотором постоянном числе позиций левее или правее него в алфавите.

Шифр XOR — это алгоритм шифрования данных с использованием исключительной дизъюнкции.

## **1.4 Организация взаимодействия параллельных потоков**

Потоки исполняются в общем адресном пространстве программы. Как результат, взаимодействие параллельных потоков можно организовать через использование общих данных, являющихся доступными для всех потоков. Наиболее простая ситуация состоит в использовании общих данных только для чтения. В случае же, когда общие данные могут изменяться несколькими потоками, необходимо блокировать участки кода, в которых происходит запись или чтение в общую переменную.

## **Вывод**

В данном разделе были рассмотрены основы конвейерной обработки, алгоритмы, которые лягут в основу конвейеров и организация взаимодействия параллельных потоков.

## 2 Конструкторская часть

### 2.1 Описание используемых структур

При реализации алгоритмов будут использованы нижеприведенные структуры.

- Заявка — обрабатываемый элемент, хранящий время входа и выхода с каждого конвейера, состояние заявки до и после обработки, время обработки и обрабатываемый элемент (строка). Также в структуре заявки находится логический флаг, для проверки является ли эта заявка последней.
- Очередь — основной компонент конвейера и линейной обработки. Здесь хранятся элементы в ожидании обработки. Очередь является разделяемым ресурсом: с ней работают различные потоки.
- Конвейер — реализует обработку элементов, хранение очереди, может манипулировать элементами.

### 2.2 Описание алгоритмов

Всего будет создано три конвейерные ленты.

На рисунках 2.1, 2.2 – 2.5 приведены схемы линейной и конвейерной обработки соответственно. Самой последней в начальную очередь добавляется особая заявка, у которой поднят флаг, отвечающий за то, то она последняя. При работе алгоритмов все заявки проходят проверку на то поднят данный флаг или нет, при положительном результате текущий этап считается завершенным

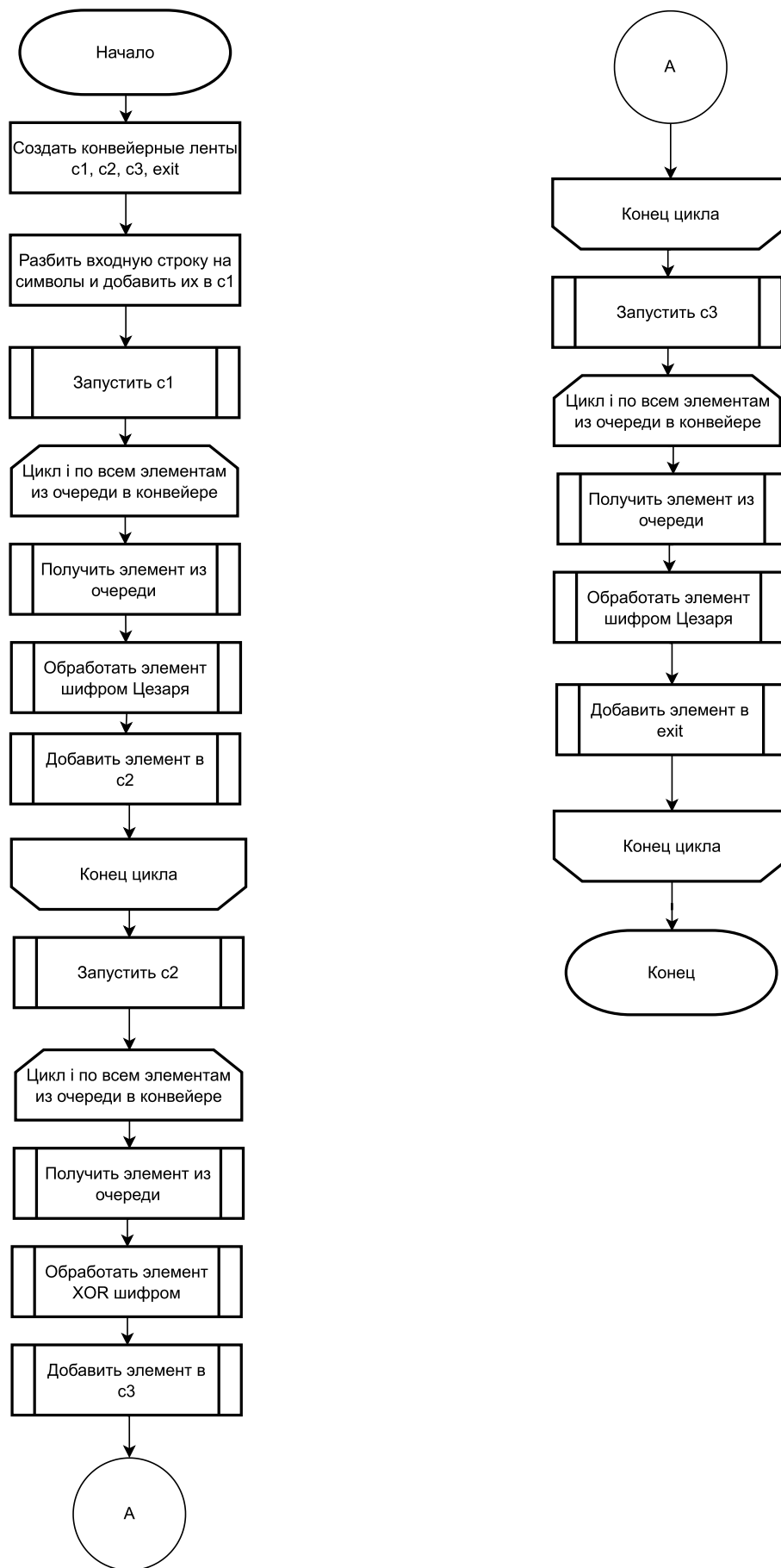


Рис. 2.1 – Схема линейной обработки

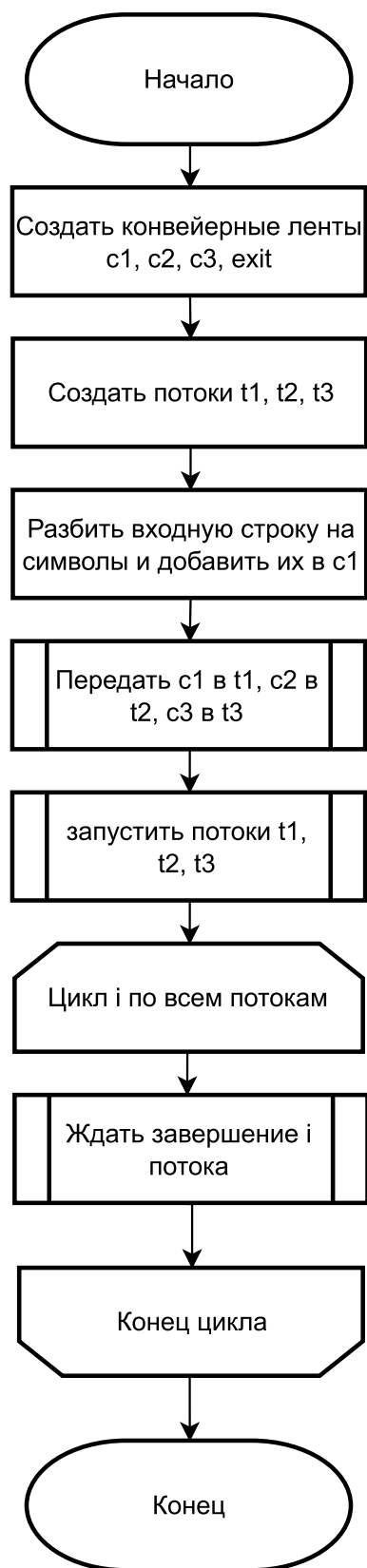


Рис. 2.2 – Схема параллельной конвейерной обработки



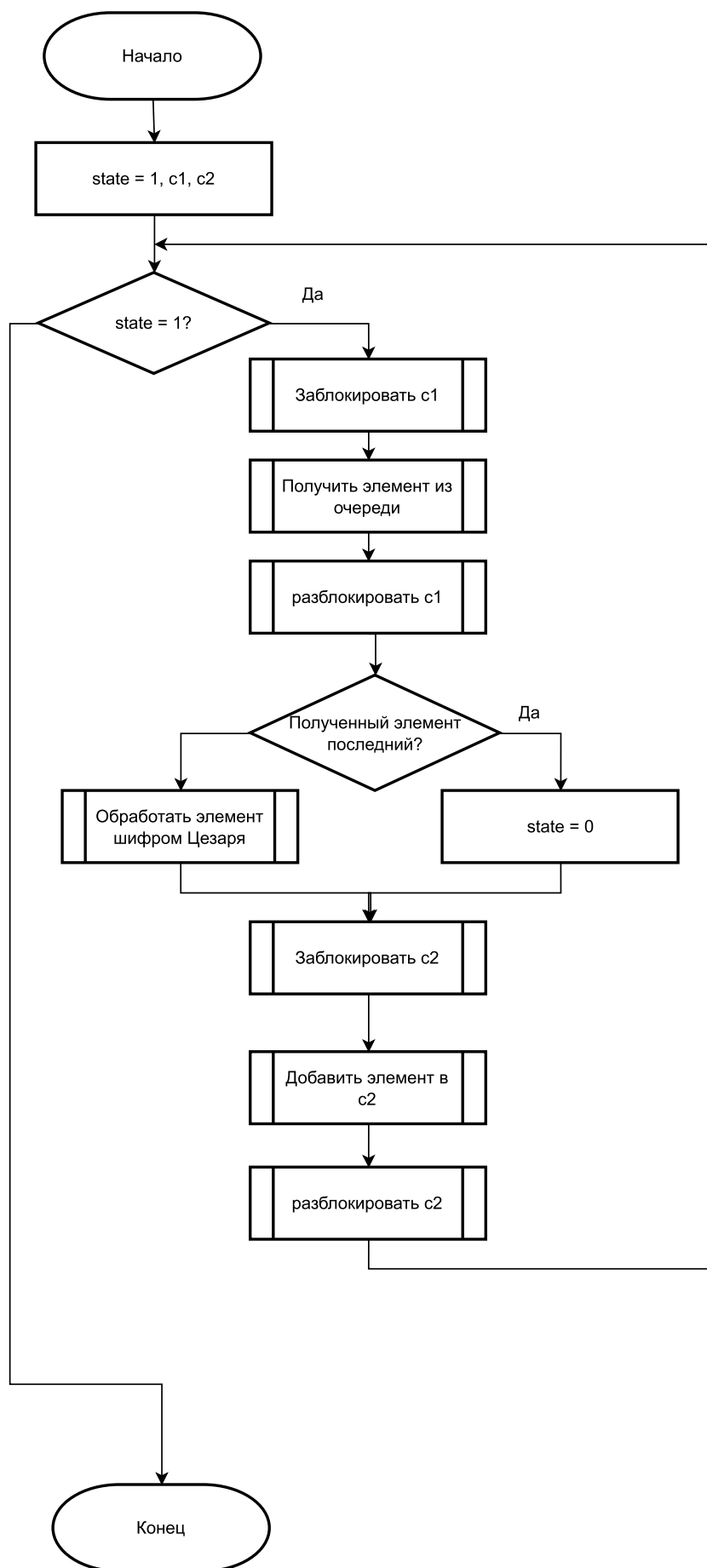


Рис. 2.3 – Схема 1 потока параллельной конвейерной обработки

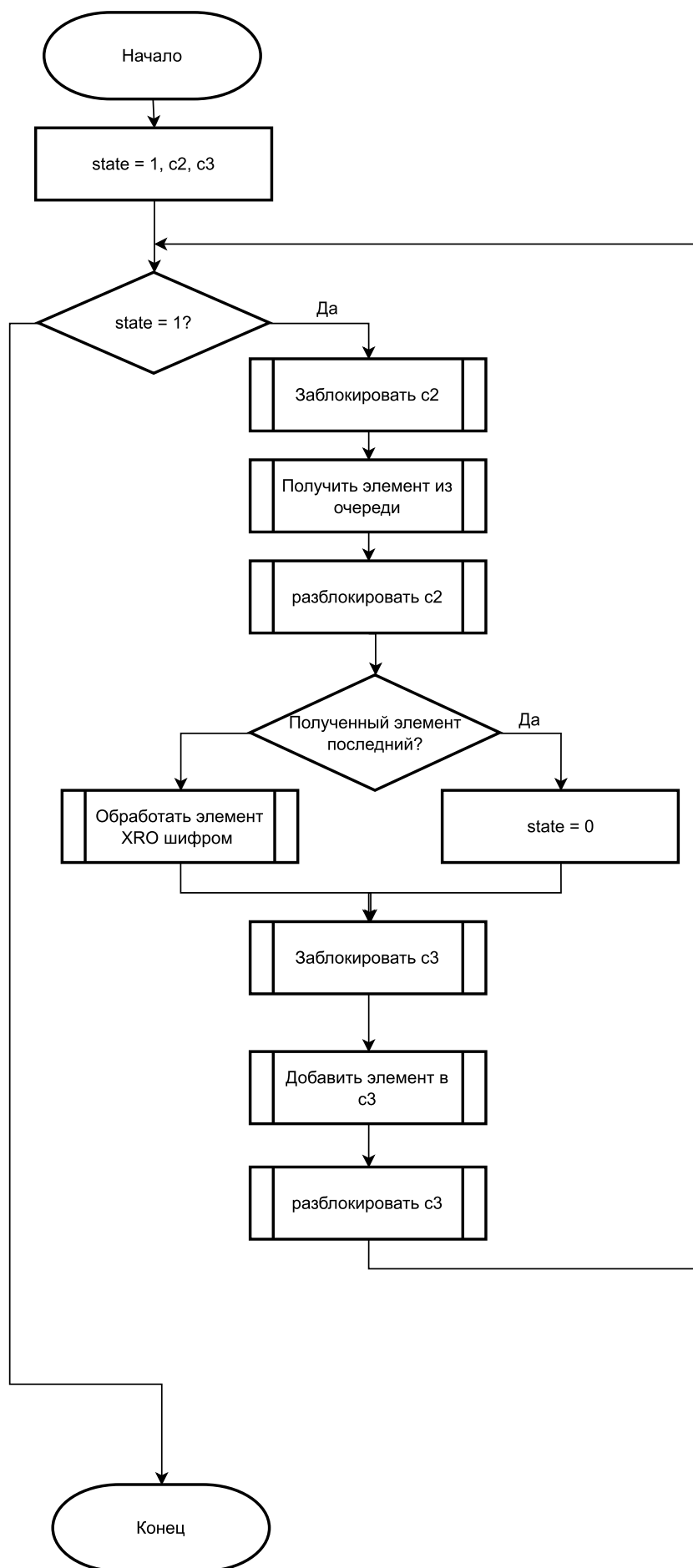


Рис. 2.4 – Схема 2 потока параллельной конвейерной обработки

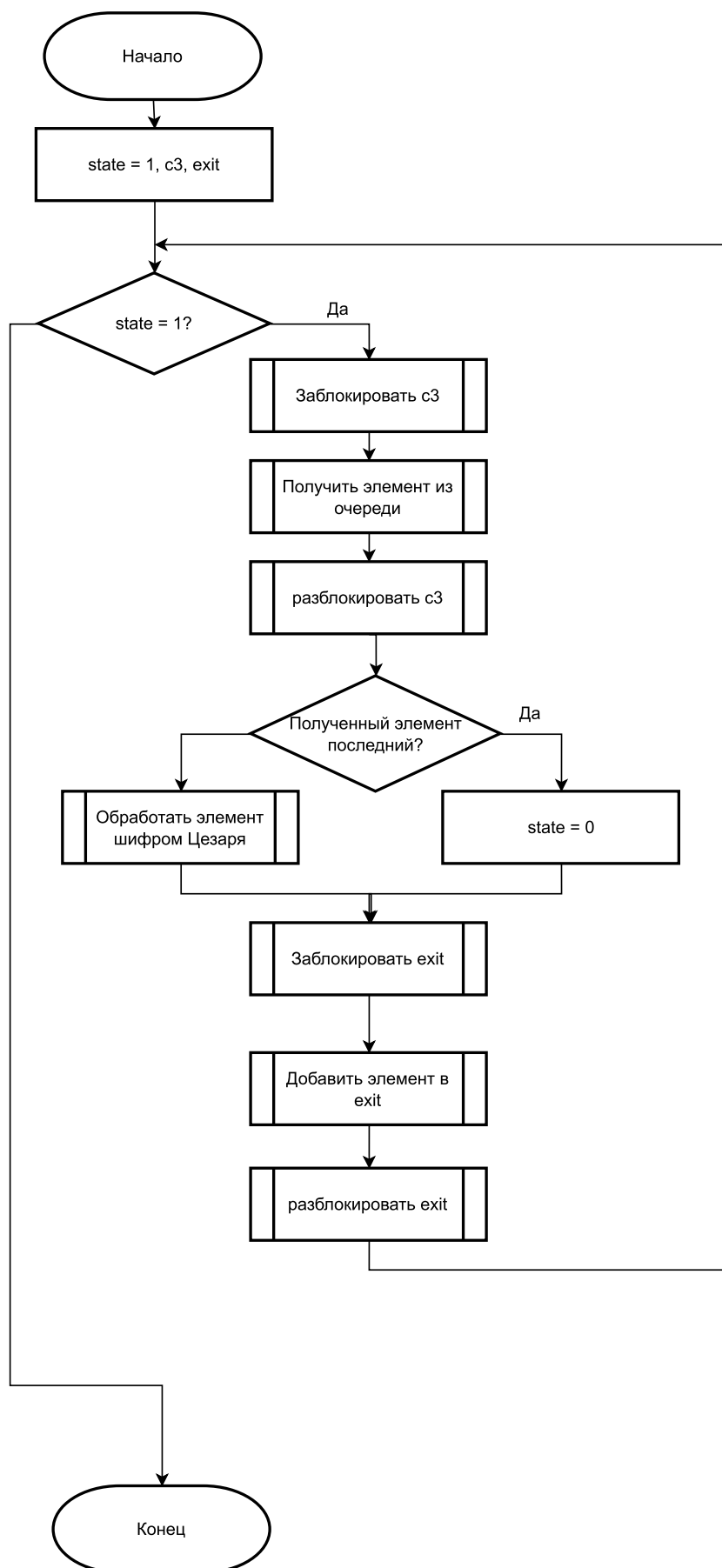


Рис. 2.5 – Схема 3 потока параллельной конвейерной обработки

## **Вывод**

В данном разделе были описаны используемые структуры и приведены схемы алгоритмов.

## 3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги алгоритмов.

### 3.1 Требования к программному обеспечению

Программа должна выводить журнал отладки для всех заявок и всех видов организации конвейерной обработки.

### 3.2 Средства реализации

В данной работе для реализации был выбран язык программирования *C#*. В текущей лабораторной работе требуется замерить время пребывания заявок в очередях. Для этого будет использована структура *DateTime*, предоставляющая текущее время. Использовать структуру приходится дважды, затем из конечного времени нужно вычесть начальное, чтобы получить результат. Так же требуется замерить суммарное время работы конвейеров для каждого из вариантов. Время работы было замерено с помощью класса *Stopwatch* из библиотеки *System.Diagnostics*. Для визуализации результатов использовался язык *Python*.

## 3.3 Сведения о модулях программы

Программа состоит из следующих модулей:

- класс *Conver* реализует конвейер;
- класс *Ask* реализует заявку;
- класс *Crypto* реализует алгоритмы шифрования;
- класс *Formatter* реализует вывод состояний на экран.

## 3.4 Реализация алгоритмов

В листингах 3.1–3.5 представлены реализации алгоритмов.

Листинг 3.1 — Реализация конвейера

```
1 class Conver
2 {
3     ConsoleApp1.Action a;
4     int id;
5     public Queue<Ask> nextState;
6     public Queue<Ask> queue;
7     int key;
8     public Conver(ConsoleApp1.Action a, int key, int id)
9     {
10         this.a = a;
11         queue = new Queue<Ask>();
12         this.key = key;
13         this.id = id;
14     }
15     public Conver(ConsoleApp1.Action a, int key, Queue<Ask> q, int
16         id) : this(a, key, id)
17     {
18         queue = q;
19     }
20     public void Start()
21     {
22         state s = state.ok;
```

```

22     while(s!=state.finish)
23     {
24         s = Process();
25     }
26 }
27 public state Process()
28 {
29     Ask elem = null;
30     bool e = false;
31     lock (queue)
32     {
33         if (queue.Count > 0)
34             e = queue.TryDequeue(out elem);
35     }
36     if (e && elem.last)
37     {
38         elem.out_time[id] = DateTime.Now.Ticks;
39         elem.in_time[id + 1] = DateTime.Now.Ticks;
40         lock (nextState)
41             { nextState.Enqueue(elem); }
42         return state.finish;
43     }
44     if (e)
45     {
46         elem.out_time[id] = DateTime.Now.Ticks;
47         long s = DateTime.Now.Ticks;
48         elem.elem = a.Invoke(elem.elem, key);
49         elem.work_time[id] = DateTime.Now.Ticks - s;
50         elem.state[id+1] = elem.elem;
51         elem.in_time[id + 1] = DateTime.Now.Ticks;
52         lock (nextState)
53             { nextState.Enqueue(elem); }
54     }
55     return state.ok;
56 }
57 public void StartSerial()
58 {
59     Ask elem = queue.Dequeue();
60     elem.out_time[id] = DateTime.Now.Ticks;
61     if (elem.last)

```

```

62     return ;
63     long s = DateTime.Now.Ticks ;
64     elem.elem = a.Invoke(elem.elem , key) ;
65     elem.work_time[id] = DateTime.Now.Ticks - s ;
66     elem.state[id + 1] = elem.elem ;
67     elem.in_time[id + 1] = DateTime.Now.Ticks ;
68     nextState.Enqueue(elem) ;
69 }
70 }

```

Листинг 3.2 — Реализация линейной обработки данных

```

1 void SerialCode ()
2 {
3     Conver[] c = new Conver[3];
4     c[0] = new Conver(new ConsoleApp1.Action(Crypto.CodeEncode)
5         ,300, q, 0);
6     c[1] = new Conver(new ConsoleApp1.Action(Crypto.Cipher) , 400,
7         1);
8     c[2] = new Conver(new ConsoleApp1.Action(Crypto.CodeEncode) ,
9         500, 2);
10    c[0].nextState = c[1].queue;
11    c[1].nextState = c[2].queue;
12    c[2].nextState = exit;
13    c[0].Start();
14    c[1].Start();
15    c[2].Start();
16 }

```

Листинг 3.3 — Реализация параллельной обработки данных

```

1 void ParalCode ()
2 {
3     Thread[] t = new Thread[3];
4     Conver[] c = new Conver[3];
5     c[0] = new Conver(new ConsoleApp1.Action(Crypto.CodeEncode)
6         ,300, q, 0);
7     c[1] = new Conver(new ConsoleApp1.Action(Crypto.Cipher) , 400,
8         1);

```



```

7      c[2] = new Conver(new ConsoleApp1.Action(Crypto.CodeEncode),
      500, 2);
8      c[0].nextState = c[1].queue;
9      c[1].nextState = c[2].queue;
10     c[2].nextState = exit;
11     for (int i = 0; i < 3; i++)
12     {
13         t[i] = new Thread(c[i].Start);
14         t[i].Start();
15     }
16     foreach (Thread thread in t)
17     {
18         thread.Join();
19     }
20 }

```

#### Листинг 3.4 — Реализация алгоритмов шифрования

```

1      static class Crypto
2      {
3          const string alfabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
4          static public char CodeEncode(char text, int k)
5          {
6              var fullAlfabet = alfabet.ToLower(); // + alfabet.ToLower
              ();
7              var letterQty = fullAlfabet.Length;
8              char retVal = '\0';
9
10             var index = fullAlfabet.IndexOf(text);
11             if (index < 0)
12             {
13                 retVal = text;
14             }
15             else
16             {
17                 var codeIndex = (letterQty + index + k) % letterQty;
18                 retVal = fullAlfabet[codeIndex];
19             }
20             return retVal;
21     }

```

```

22     static char GetRandomKey(int k)
23     {
24         char gamma = '\0';
25         var rnd = new Random(k);
26         gamma = (char)rnd.Next(97, 123);
27         return gamma;
28     }
29     static public char Cipher(char text, int key)
30     {
31         var currentKey = GetRandomKey(key);
32         char res = '\0';
33         res = ((char)(text ^ key));
34         return res;
35     }
36 }

```

Листинг 3.5 — Реализация структуры заявки

```

1     class Ask
2     {
3         public long[] in_time, out_time, work_time;
4         public char[] state;
5         public char elem;
6         public bool last;
7         public Ask(int n, char elem, bool last = false)
8         {
9             in_time = new long[n];
10            out_time = new long[n];
11            work_time = new long[n];
12            state = new char[n];
13            this.elem = elem;
14            this.last = last;
15        }
16    }

```

## 3.5 Функциональное тестирование

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы конвейерной обработки. Тесты для всех реализаций алгоритмов пройдены успешно.

Таблица 3.1 — Функциональные тесты

Входная строка	Выходная строка
Пустая строка	Сообщение об ошибке
a	вывод журнала отладки
asdfghj	вывод журнала отладки
qwerty	вывод журнала отладки

## Вывод

В данном разделе были рассмотрены листинги используемых алгоритмов, проведено функциональное тестирование.

## **4 Исследовательская часть**

В данном разделе будут приведены примеры работы программы, а также проведен сравнительный анализ процессорного времени работы реализаций алгоритмов при различных ситуациях на основе полученных данных.

### **4.1 Технические характеристики**

Технические характеристики устройства, на котором выполнялись замеры времени, представлены далее:

- операционная система Windows 11 Pro Версия 22H2 (22621.674) [1];
- память 16 ГБ;
- процессор 11th Gen Intel(R) Core(TM) i5-11400 2.59 ГГц [2], 6 физических и 12 логических ядер.

При тестировании компьютер был включен в сеть электропитания. Во время замеров процессорного времени устройство было нагружено только встроенными приложениями окружения, а также системой тестирования.

### **4.2 Демонстрация работы программы**

На рисунке 4.1 представлен результат работы программы. Работающая программа выводит на экран журнал отладки для каждой заявки.

```

&&@Parallel@&&
11
00.01,
Conv 0: a -> u|in q 638053644748595506, out q 638053644748742165, wait 14; work 7
-----
Conv 1: u -> 127|in q 638053644748819498, out q 638053644748819600, wait 0; work 0
-----
Conv 2: 127 -> 127|in q 638053644748826519, out q 638053644748826545, wait 0; work 0
-----
All time wait 14
*****
Conv 0: s -> m|in q 638053644748610712, out q 638053644748819592, wait 20; work 0
-----
Conv 1: m -> 103|in q 638053644748819646, out q 638053644748826538, wait 0; work 0
-----
Conv 2: 103 -> 103|in q 638053644748826543, out q 638053644748826649, wait 0; work 0
-----
All time wait 20
*****
Conv 0: d -> x|in q 638053644748610726, out q 638053644748819648, wait 20; work 0

```

Рис. 4.1 – Пример работы программы

### 4.3 Время выполнения реализаций алгоритмов

**Входные данные:** 10000 строк, размером от 500 до 2500 символов.

В таблице ниже приведены результаты замеров времени (в тиках) для общего времени пребывания заявок в очередях.

Таблица 4.1 — Суммарное время пребывания всех заявок в очереди в тиках  $\cdot 10^{-5}$

Длина входных строк	Линейный конвейер	Параллельный конвейер
500	33102498	15664766
1000	105645693	49704767
2000	364612686	185795895
2500	536216748	276675101

На рисунке 4.2 приведены графические результаты замеров времени работы алгоритмов в зависимости от линейного размера входной строки.

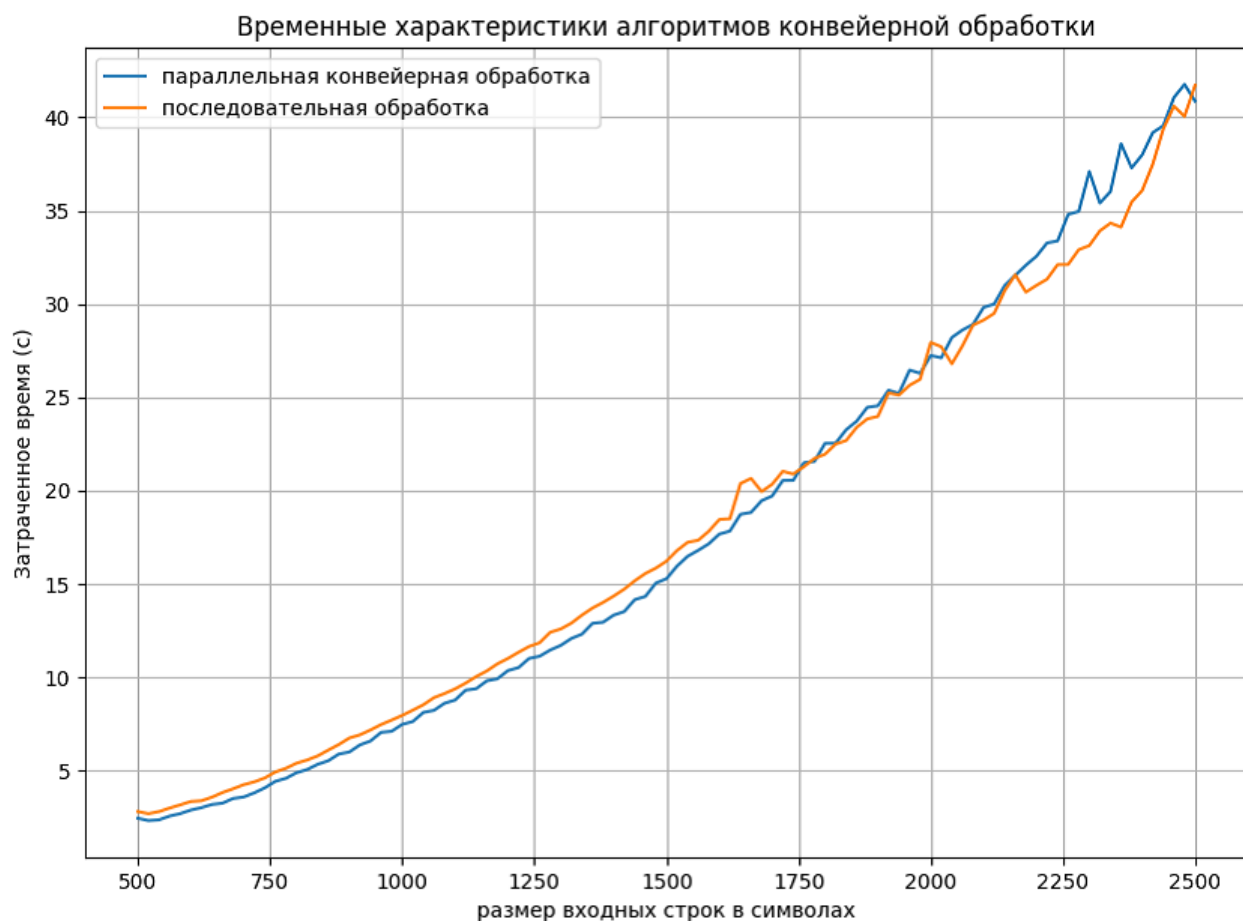


Рис. 4.2 – Время вычислений

## 4.4 Вывод

По полученным результатам исследования можно сделать вывод, что конвейер выигрывает от двух до трех раз в скорости работы и снижении времени ожидания заявок в очереди перед линейной реализацией.

# Заключение

Цель, которая была поставлена в начале лабораторной работы, была достигнута: изучены, реализованы и исследованы принципы конвейерной обработки данных.

В ходе выполнения лабораторной работы были решены все задачи:

- 1) описаны основные принципы конвейерной обработки;
- 2) описан последовательный алгоритм обработки данных;
- 3) разработаны обработчики заявок в составе конвейера и алгоритмы, выполняемые на них;
- 4) реализованы заданные алгоритмы и конвейер;
- 5) проведены замеры времени для последовательного алгоритма обработки данных и для алгоритма конвейерной обработки данных;
- 6) выполнен анализ полученных результатов;
- 7) по итогам работы составлен отчет.

В ходе проделанной работы было выявлено, что конвейерная обработка и последовательная имеют линейные затраты по времени. Однако сложность конвейера имеет меньший коэффициент роста по сравнению с линейным конвейером.

# Список использованных источников

1. *Windows 11, version 22H2 [Эл. ресурс]*. Режим доступа: <https://clck.ru/32NCXx> (дата обращения: 14.10.2022).
2. *Процессор Intel® Core™ i7 [Эл. ресурс]*. Режим доступа: <https://clck.ru/yeQa8> (дата обращения: 14.10.2022).