



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» (ИУ7)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.04 Программная инженерия

О Т Ч Е Т

по лабораторной работе № 5

Название: Разработка конвейера, работающего в параллельном
режиме

Дисциплина: Анализ алгоритмов

Студент

ИУ7-52Б

(Группа)

(Подпись, дата)

Е.В. Брянская

(И.О. Фамилия)

Преподаватель

Л.Л. Волкова

(Подпись, дата)

(И.О. Фамилия)

Москва, 2020

Оглавление

Введение	3
1 Аналитическая часть	4
1.1 Цель и задачи	4
1.2 Основные требования к алгоритмам шифрования	4
1.3 Используемые алгоритмы шифрования	5
1.3.1 Алгоритм, в основе которого лежит операция XOR	5
1.3.2 Шифр Виженера	5
1.3.3 Транспозиция	5
1.4 Общие принципы работы конвейера	6
Вывод	6
2 Конструкторская часть	7
2.1 Используемые алгоритмы шифрования	7
2.1.1 Алгоритм, в основе которого лежит операция XOR	7
2.1.2 Шифр Виженера	8
2.1.3 Транспозиция	9
2.2 Организация работы конвейера	10
2.3 Требования к ПО	14
2.4 Заготовки тестов	14
Вывод	14
3 Технологическая часть	15
3.1 Выбранный язык программирования	15
3.2 Листинг кода	15
3.3 Замеры времени	21
Вывод	21
4 Экспериментальная часть	22
4.1 Характеристики ПК	22
4.2 Лог конвейерной обработки	22
Вывод	23
Заключение	24
Список литературы	25

Введение

В этой лабораторной работе будет рассматриваться разработка конвейера, работающего в параллельном режиме. В качестве основной работы берётся шифрование строк, которое разделено на три стадии.

Конвейеризация (или конвейерная обработка) в общем случае основана на разделении подлежащей исполнению задачи на более мелкие части, называемые ступенями/стадиями, организовав передачу данных от одного этапа к следующему. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько задач [1].

Моделирование конвейерной обработки будет осуществляться с помощью потоков, на каждую ленту выделяется отдельный поток (или несколько, в зависимости от сложности решаемой задачи).

Шифрование (зашифрование) — процесс применения шифра к защищаемой информации, т.е. преобразование исходного сообщения в зашифрованное.

Дешифрование (расшифрование) — процесс, обратный шифрованию, т. е. преобразование зашифрованного сообщения в исходное.

Под **шифром** понимается совокупность методов и способов обратимого преобразования информации с целью ее защиты от несанкционированного доступа (обеспечения конфиденциальности информации).

Ключ — это минимально необходимая информация (за исключением сообщения, алфавитов и алгоритма), для шифрования и дешифрования сообщений.

Под **алгоритмом** подразумевается набор правил (инструкций), определяющих содержание и порядок операций по шифрованию и дешифрованию информации.

В данной лабораторной работе будут использоваться такие алгоритмы шифрования, как усложненный метод с использованием операции XOR, алгоритм Виженера и метод транспозиции.

1. Аналитическая часть

В этом разделе будут поставлены цель и основные задачи лабораторной работы, которые будут решаться в ходе её выполнения.

1.1. Цель и задачи

Цель данной работы: получить навык разработки конвейера, работающего в параллельном режиме.

Для достижения поставленной цели необходимо решить ряд следующих **задач**:

- 1) описать алгоритмы, на основе которых будет строиться работа конвейера;
- 2) описать алгоритм работы конвейера;
- 3) реализовать все рассмотренные алгоритмы;
- 4) получить время ожидания в очереди и полное время решения для каждой задачи;
- 5) среди найденных временных значений найти минимальное/максимальное/среднее время.

В качестве основной работы берётся шифрование строк, которое разделено на три стадии, на каждой из которых представлен свой алгоритм.

1.2. Основные требования к алгоритмам шифрования

Сюда нужно отнести следующее.

- Алгоритм должен быть надёжным, не допускать ситуации, когда ключ (если он используется) и сам принцип шифрования успешно угадывались сторонними лицами.
- Должен допускать эффективную программную реализацию.
- И быть достаточно простым для написания кода, чтобы минимизировать вероятность программных ошибок.
- Если алгоритм использует ключи, то любую случайную строку битов нужной длины, следует рассматривать в качестве возможного ключа.
- Используемый метод должен легко модифицироваться для различных уровней безопасности и удовлетворять как минимальным, так и максимальным требованиям.

1.3. Используемые алгоритмы шифрования

1.3.1 Алгоритм, в основе которого лежит операция XOR

Один из самых простых алгоритмов шифрования. Он основан на применении бинарной логической операции исключающего или (XOR).

Операция XOR обладает симметричностью. Это значит, что если зашифровать одну и ту же строку 2 раза с одним и тем же ключом, то в результате получается эта же строка без изменений.

1.3.2 Шифр Виженера

Этот метод используется для шифрования буквенного текста с использованием ключевого слова. В его основе лежит алгоритм Цезаря, но в отличие от него, он более надежный и безопасный.

Шифр можно записать с помощью формулы 1.1.

$$c_i = (m_i + k_i) \bmod n, \quad (1.1)$$

где m_i – код i -ой буквы строки, которую нужно зашифровать, k_i – код буквы ключа, n – количество букв в алфавите, c_i – итоговое значение кода зашифрованной буквы, \bmod – операция получения остатка от деления. Далее по коду символа определяется соответствующая ему буква [2].

1.3.3 Транспозиция

Этот алгоритм основан на перестановке символов в строке по какому-либо правилу. Для усложнения метода используется двойная перестановка, т.е. после первого прохода по строке, осуществляется второй проход, также меняющий местами символы, но уже по другому принципу.

Таким образом, не зная правил, по которым производилось шифрование достаточно сложно подобрать нужные. Задача усложняется тем, что неизвестно сколько и какие методы используются в конкретном случае [3]

1.4. Общие принципы работы конвейера

Пусть конвейер состоит из трёх лент. Тогда будем симулировать каждую из них, потоком, причём, на каждую ленту выделено по одному потоку. Под потоком подразумевается непрерывная часть кода процесса, которая может выполняться с другими частями выполняемой программы.

Пусть существует план – обработать N объектов. Рабочий поток живёт до тех пор, пока не выполнит весь план.

Главный поток запускает рабочие потоки и выдаёт задачи (объекты) на обработку первому потоку. После того, как задача обрабатывается на первой ленте, она передаётся на вторую, в это время как, на первую поступает другая задача. Аналогично организуется работа на втором и третьем потоках. Процессы на лентах выполняются параллельно.

Когда все задачи будут последовательно обработаны на трёх лентах конвейера, главный поток соберёт статистику и выведет её в интерфейс, либо файл.

Перед тем, как пройти стадию обработки на какой-либо ленте, каждая задача попадает в очередь из таких же задач. Максимально полезная работа контейнера возникает тогда, когда загружены все потоки, и время простоя минимально.

Вывод

Был рассмотрен общий принцип работы конвейера, а также алгоритмы шифрования на каждой из его лент.

2. Конструкторская часть

Пусть на вход подаётся последовательность символов латинского алфавита. Рассмотрим работу конвейера для шифрования этих данных.

2.1. Используемые алгоритмы шифрования

2.1.1 Алгоритм, в основе которого лежит операция XOR

На вход подаётся строка и ключ. Этот алгоритм шифрования разбит на две части: сначала производится операция XOR каждой буквы строки с первой буквой ключа, затем с каждой третьей буквой дополнительно производится та же операция, только уже со второй буквой ключа (при условии, что длина ключа больше 1).

Схема алгоритма представлена на Рис.2.1.

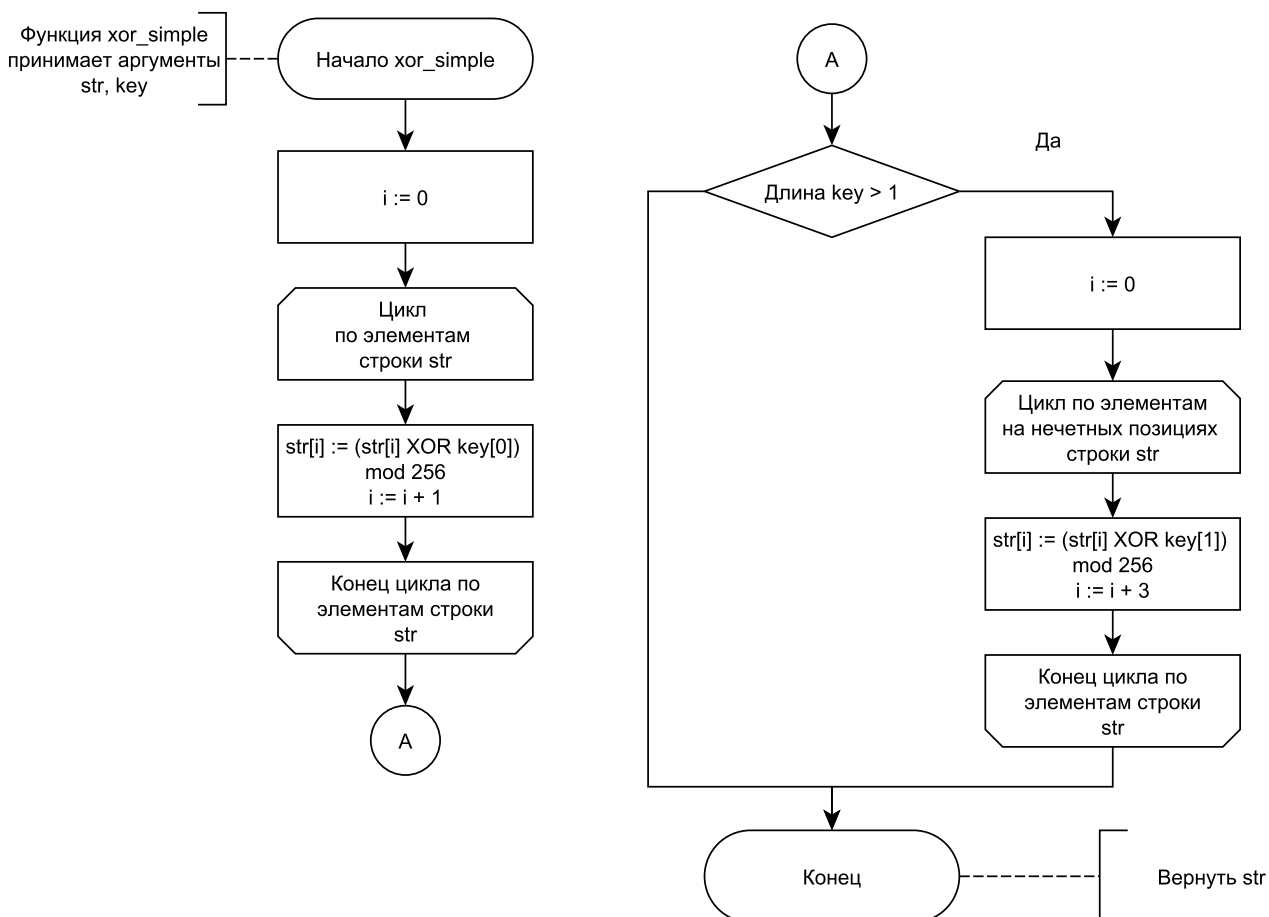


Рис. 2.1 — Алгоритм, в основе которого лежит операция XOR

2.1.2 Шифр Виженера

На вход тоже подаётся строка и ключ. Реализовывается проход по всем элементам строки, и согласно формуле 1.1, код текущего символа складывается с кодом соответствующего ему символа ключа, и затем по полученному коду определяется сам символ.

Если ключ короче исходной строки, то он используется циклически, т.е. после последнего символа ключа вновь происходит переход к первому.

Схема алгоритма представлена на Рис.2.2.

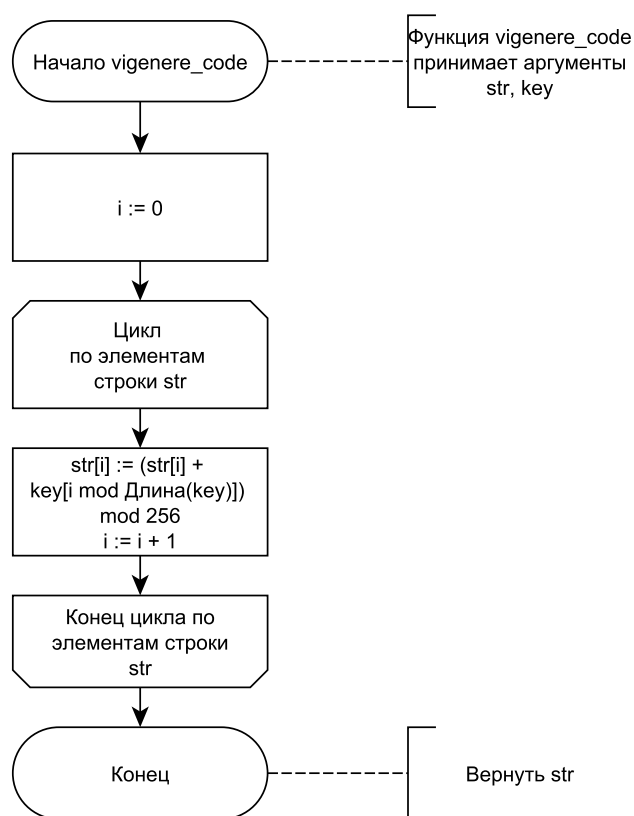


Рис. 2.2 — Шифр Виженера

2.1.3 Транспозиция

В этом алгоритме используется только символьная строка. В рассматриваемом методе используется двойная перестановка. Сначала переставляются элементы попарно, а затем все элементы первой половины строки, стоящие на нечетных позициях, меняются местами с симметричными относительно середины строки элементами.

Схема алгоритма представлена на Рис.2.3.

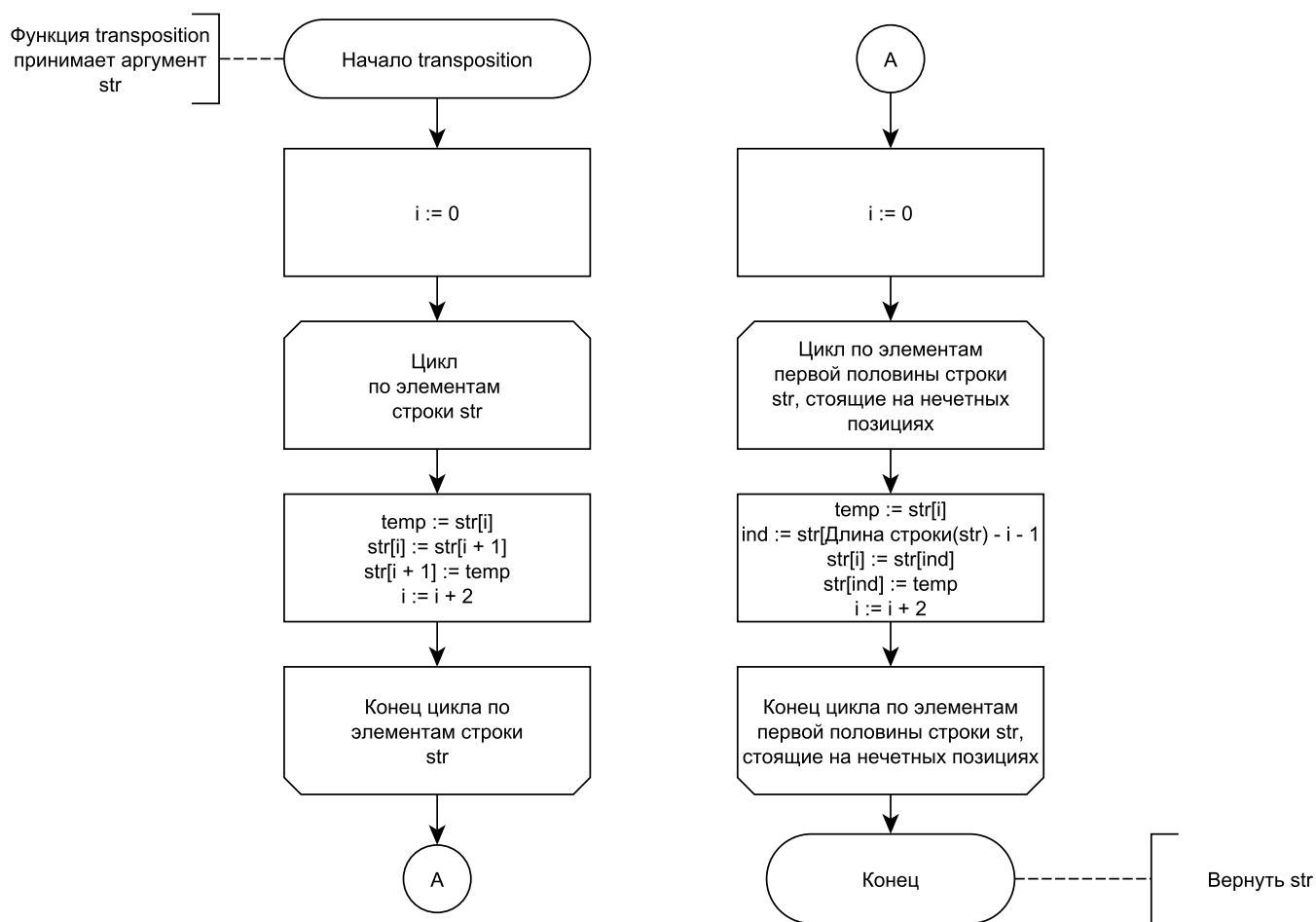


Рис. 2.3 — Транспозиция

2.2. Организация работы конвейера

Как упоминалось выше, рассматриваемый конвейер состоит из трёх лент, каждая из которых обрабатывается своим потоком. Перед тем, как поступить на какую-либо ленту задача попадает в очередь и ожидает дальнейшей обработки.

Первая очередь заполняется заранее, число задач указывается пользователем. Если ещё не все задачи обработаны, а очередь на какую-либо стадию обработки пустая, то лента уходит в режим ожидания следующего объекта.

Время обработки на лентах примерно одинаковое, и выполняется условие 2.1.

$$t_{processing} \gg t_{dispatching} \quad (2.1)$$

Работа конвейера представлена на **схемах**: главный поток – Рис.2.4, рабочие потоки – Рис.2.5 (1-ая лента), Рис.2.6 (2-ая лента), Рис.2.7 (3-я лента).

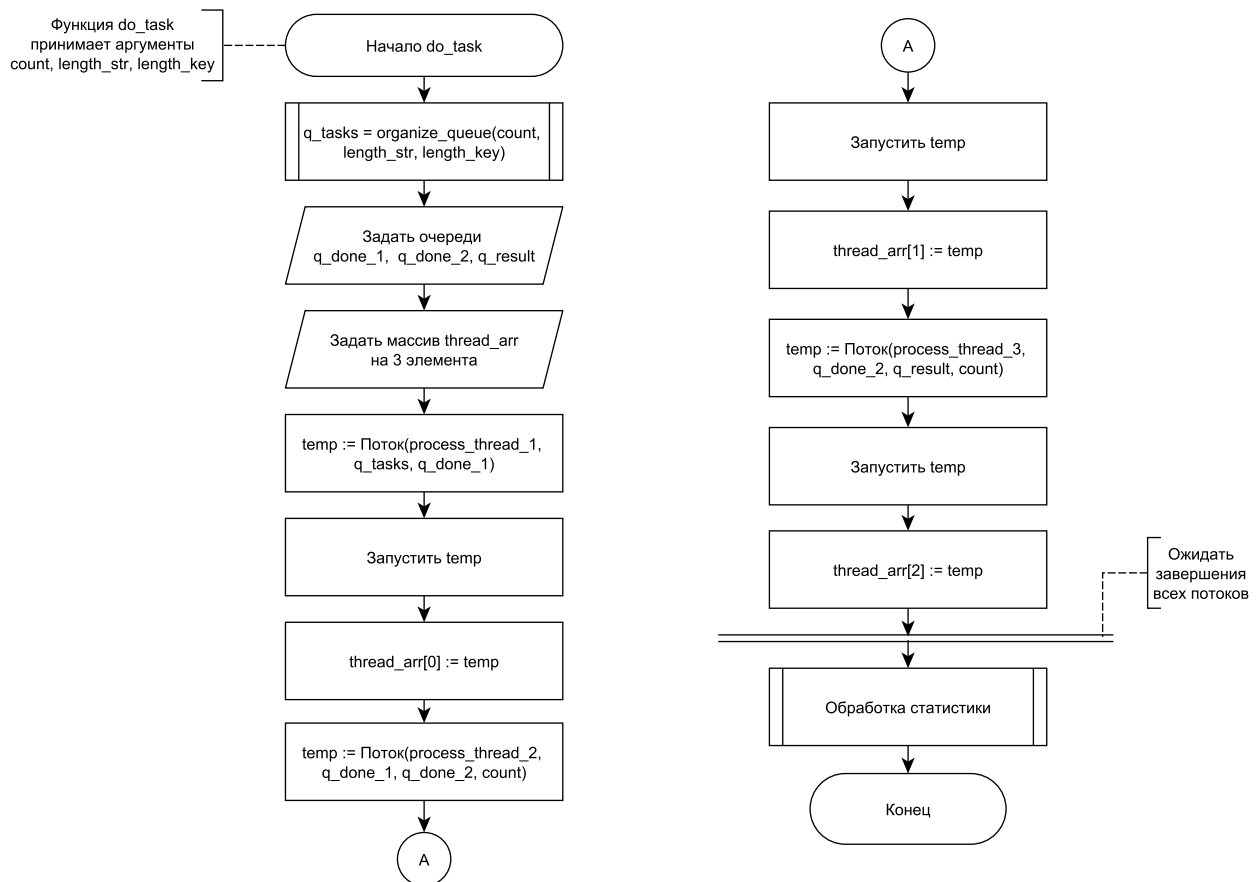


Рис. 2.4 — Организация работы конвейера. Главный поток

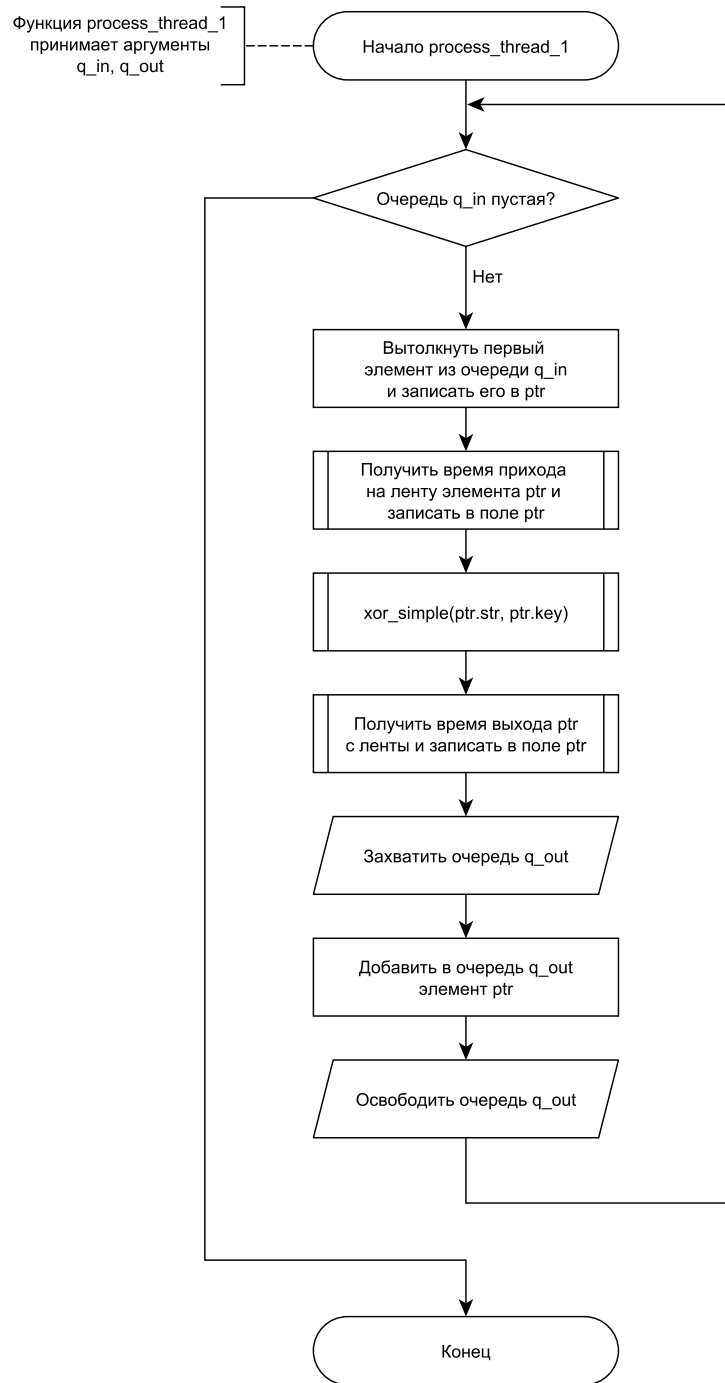


Рис. 2.5 — Организация работы конвейера. Лента 1, рабочий поток

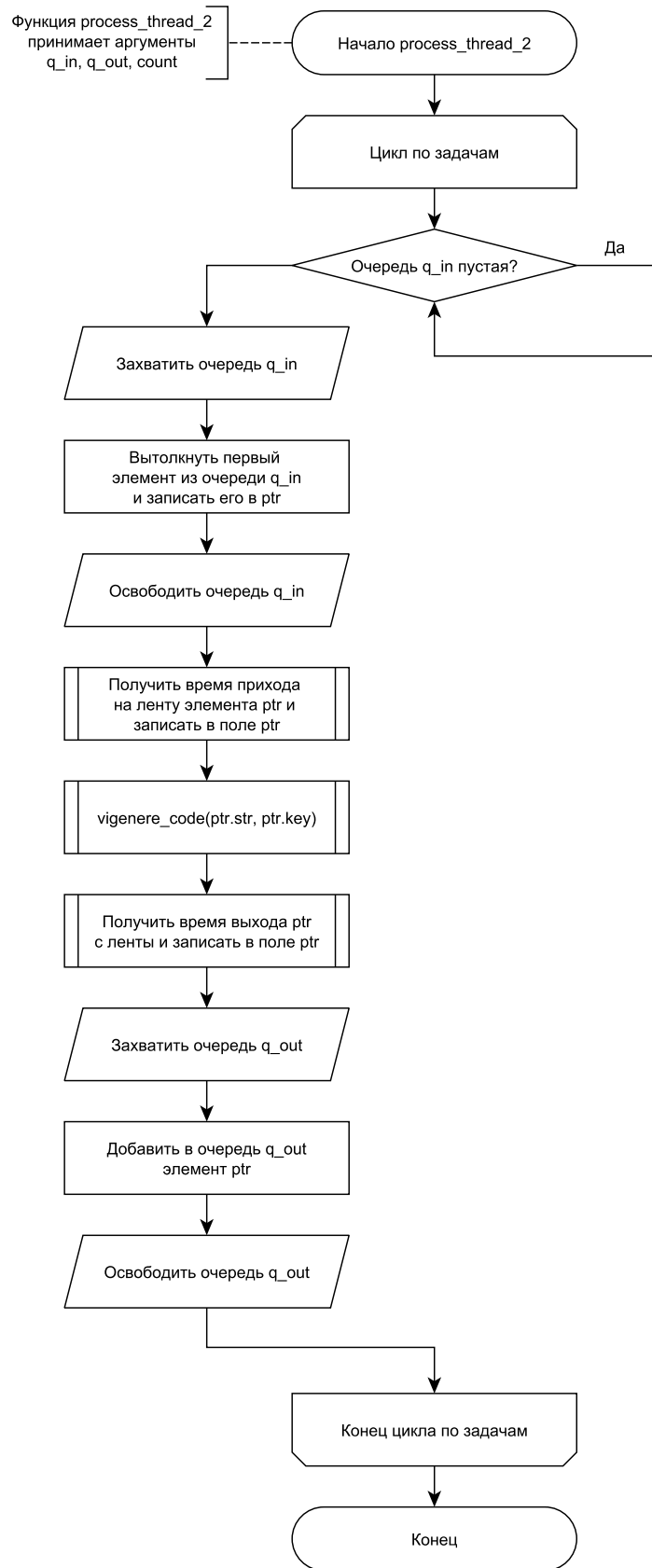


Рис. 2.6 — Организация работы конвейера. Лента 2, рабочий поток

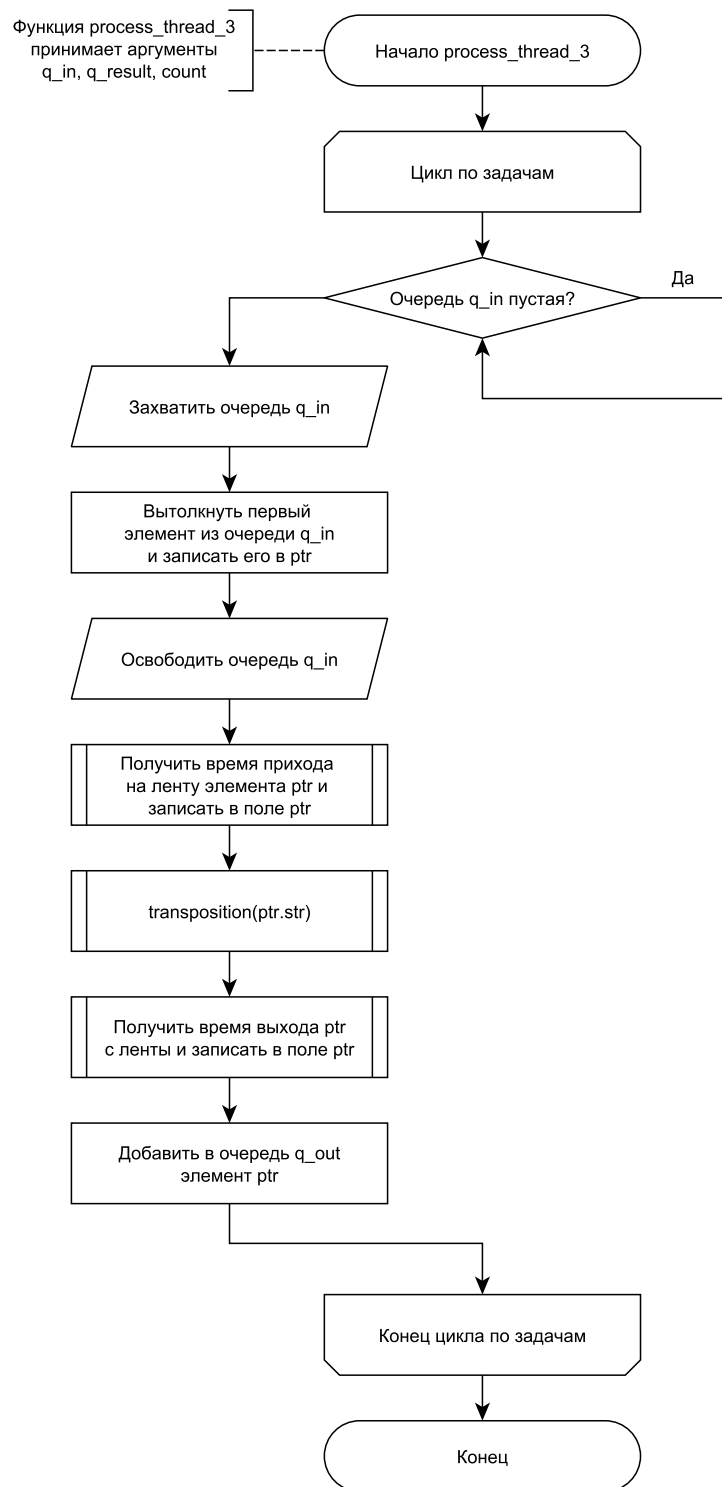


Рис. 2.7 — Организация работы конвейера. Лента 3, рабочий поток

2.3. Требования к ПО

Для корректной работы алгоритмов и проведения тестов необходимо выполнить следующее.

- Обеспечить возможность ввода количества задач через консоль.
- В случае ввода некорректных данных вывести соответствующее сообщение. Программа не должна аварийно завершаться.
- Каждая лента должна работать в своём потоке.
- Обеспечить возможность отслеживания времени, когда каждая задача была подана на конкретную ленту конвейера и когда закончилась её обработка.
- Реализовать возможность вывода на экран максимальных, минимальных и средних значений времени обработки целой задачи и ожидания в очереди, и итоговое время работы системы.

2.4. Заготовки тестов

При проверке на корректность работы конвейера необходимо провести следующие тесты:

- количество задач меньше 10;
- количество задач больше 30.

Вывод

В этом разделе разобраны основные принципы выбранных алгоритмов шифрования, построены схемы. Также описана работа конвейера, его составных частей, приведены соответствующие схемы. Приведены требования к программному обеспечению и написаны заготовки тестов, которые будут использоваться в дальнейшем.

3. Технологическая часть

В данном разделе будут приведены листинги функций разрабатываемых алгоритмов шифрования и работы конвейера.

3.1. Выбранный язык программирования

Для выполнения этой лабораторной работы был выбран язык программирования C++, так как есть большой навык работы с ним и с подключаемыми библиотеками, которые также использовались для проведения тестирования и замеров. Для реализации потоков использовались библиотеки *thread* [4], *mutex* [5], для очередей – *queue* [6].

Использованная среда разработки - Visual Studio [7].

3.2. Листинг кода

Ниже представлены Листинги 3.1, 3.2, 3.3 функций, реализующих алгоритмы шифрования символьных строк. На Листингах 3.4, 3.5 приведены функции, работающие с конвейером.

Листинг 3.1 — Алгоритм с использованием операции XOR

```
1 void xor_simple(string& str, string key)
2 {
3     for (size_t i = 0; i < str.length(); i++)
4         str[i] = (str[i] ^ key[0]) % 256;
5
6     if (key.length() > 1)
7         for (size_t i = 0; i < str.length(); i += 3)
8             str[i] = (str[i] ^ key[1]) % 256;
9 }
```

Листинг 3.2 — Шифр Виженера

```
1 void vigenere_code(string& str, string key)
2 {
3     for (size_t i = 0; i < str.length(); i++)
4         str[i] = (str[i] + key[i % key.length()]) % 256;
5 }
```

Листинг 3.3 — Транспозиция

```
1 void transposition(string& str)
2 {
```

```

3  char temp;
4
5  for (size_t i = 0; i < str.length() - 1; i += 2)
6  {
7      temp = str[i];
8      str[i] = str[i + 1];
9      str[i + 1] = temp;
10 }
11
12 for (size_t i = 0; i < str.length() / 2; i += 2)
13 {
14     temp = str[i];
15     str[i] = str[str.length() - i - 1];
16     str[str.length() - i - 1] = temp;
17 }
18 }

```

Листинг 3.4 — Работа с конвейером

```

1  queue<shared_ptr<task>> organize_queue(int count, int len_str, int
    len_key)
2  {
3      queue<shared_ptr<task>> q_tasks;
4
5      for (int i = 0; i < count; i++)
6      {
7          string str = generate_string(len_str);
8          string key_letter = generate_string(len_key);
9
10         shared_ptr<task> ptr(new task);
11         ptr->str = str;
12         ptr->key = key_letter;
13
14         q_tasks.push(ptr);
15     }
16
17     return q_tasks;
18 }
19
20 void do_task(int count)
21 {

```



```

22  int length_str = 1000000;
23  int length_key = 20;
24  double t, cur_process, cur_delay, time_start, time_end;
25  double max_process = -1, min_process = 1000000, avg_process = 0,
26  max_delay = -1, min_delay = 1000000, avg_delay = 0;
27
28  queue<shared_ptr<task>> q_tasks = organize_queue(count, length_str,
29  length_key);
30
31  queue<shared_ptr<task>> q_done_1, q_done_2, q_result;
32
33  shared_ptr<task> temp;
34
35  vector<thread> thread_arr;
36
37  start_measuring();
38
39  thread_arr.push_back(thread(process_thread_1, ref(q_tasks), ref(
40  q_done_1)));
41  thread_arr.push_back(thread(process_thread_2, ref(q_done_1), ref(
42  q_done_2), count));
43  thread_arr.push_back(thread(process_thread_3, ref(q_done_2), ref(
44  q_result), count));
45
46  for (int i = 0; i < thread_arr.size(); i++)
47      thread_arr[i].join();
48
49  cout << endl << endl << " #";
50  cout << "1st process (in/out) ";
51  cout << " 2nd process (in/out) ";
52  cout << " 3d process (in/out)" << endl;
53
54  for (int i = 0; i < count; i++)
55  {
56      temp = q_result.front();
57      if (i == 0)
58          time_start = temp->time_in_1;
59      else if (i == count - 1)
60          time_end = temp->time_out_3;
61
62      cout << i + 1;

```

```

58     cout << temp->time_in_1 << " - " << temp->time_out_1;
59     cout << temp->time_in_2 << " - " << temp->time_out_2;
60     cout << temp->time_in_3 << " - " << temp->time_out_3 << endl;
61     q_result.pop();
62
63     cur_process = temp->time_out_3 - temp->time_in_1;
64     if (cur_process > max_process)
65         max_process = cur_process;
66     if (cur_process < min_process)
67         min_process = cur_process;
68
69     avg_process += cur_process;
70
71     cur_delay = temp->time_in_2 - temp->time_out_1;
72     if (cur_delay > max_delay)
73         max_delay = cur_delay;
74     if (cur_delay < min_delay)
75         min_delay = cur_delay;
76
77     avg_delay += cur_delay;
78
79     cur_delay = temp->time_in_3 - temp->time_out_2;
80     if (cur_delay > max_delay)
81         max_delay = cur_delay;
82     if (cur_delay < min_delay)
83         min_delay = cur_delay;
84
85     avg_delay += cur_delay;
86 }
87 avg_process /= count;
88 avg_delay /= (count / 2);
89
90 cout << "Min";
91 cout << "Max";
92 cout << "Avg" << endl;
93
94 cout << "Whole task";
95 cout << min_process;
96 cout << max_process;
97 cout << avg_process << endl;

```

```

98
99     cout << "Delay";
100     cout << min_delay;
101     cout << max_delay;
102     cout << avg_delay << endl << endl;
103
104     cout << "General time: " << time_end - time_start << endl;
105 }

```

Листинг 3.5 — Ленты конвейера

```

1  mutex set_mutex;
2
3  void process_thread_1(queue<shared_ptr<task>>& q_in, queue<shared_ptr<
4      task>>& q_out)
5  {
6      while (!q_in.empty()) {
7          shared_ptr<task> ptr(q_in.front());
8          q_in.pop();
9
10         double t = get_measured();
11         ptr->time_in_1 = t;
12
13         xor_simple(ptr->str, ptr->key);
14
15         t = get_measured();
16         ptr->time_out_1 = t;
17
18         set_mutex.lock();
19         q_out.push(ptr);
20         set_mutex.unlock();
21     }
22 }
23
24 void process_thread_2(queue<shared_ptr<task>>& q_in, queue<shared_ptr<
25     task>>& q_out, int count)
26 {
27     for (int i = 0; i < count; i++)
28     {
29         while (q_in.empty()) {}
30     }
31 }

```

```

29     shared_ptr<task> ptr(q_in.front());
30     set_mutex.lock();
31     q_in.pop();
32     set_mutex.unlock();
33
34     double t = get_measured();
35     ptr->time_in_2 = t;
36
37     vigenere_code(ptr->str, ptr->key);
38
39     t = get_measured();
40     ptr->time_out_2 = t;
41
42     set_mutex.lock();
43     q_out.push(ptr);
44     set_mutex.unlock();
45 }
46 }
47
48 void process_thread_3(queue<shared_ptr<task>>& q_in, queue<shared_ptr<
    task>>& q_out, int count)
49 {
50     for (int i = 0; i < count; i++)
51     {
52         while (q_in.empty()) {}
53
54         shared_ptr<task> ptr(q_in.front());
55         set_mutex.lock();
56         q_in.pop();
57         set_mutex.unlock();
58
59         double t = get_measured();
60         ptr->time_in_3 = t;
61         transposition(ptr->str);
62         t = get_measured();
63         ptr->time_out_3 = t;
64
65         q_out.push(ptr);
66     }
67 }

```

3.3. Замеры времени

Для того, чтобы отследить в какой момент времени конкретная задача была подана на ленту и когда закончилась её обработка, были написаны специальные функции (Листинг 3.6), использующие метод `QueryPerformanceCounter` библиотеки `windows.h` [8].

Листинг 3.6 — Работа со временем

```
1 double PCFreq = 0.0;
2 __int64 CounterStart = 0;
3
4
5 void start_measuring()
6 {
7     LARGE_INTEGER li;
8     QueryPerformanceFrequency(&li);
9
10    PCFreq = double(li.QuadPart) / 1000;
11
12    QueryPerformanceCounter(&li);
13    CounterStart = li.QuadPart;
14 }
15
16 double get_measured()
17 {
18     LARGE_INTEGER li;
19     QueryPerformanceCounter(&li);
20
21     return double(li.QuadPart - CounterStart) / PCFreq;
22 }
```

Вывод

Были разработаны функции, реализующие выбранные алгоритмы шифрования символьных строк, также алгоритм работы конвейера, удовлетворяющий всем требованиям, и приведены листинги кода каждой из них.

4. Экспериментальная часть

В этом разделе будут представлены результаты, которые демонстрирует разработанный конвейер, а именно, какое максимальное, минимальное и среднее время тратится на обработку строк длиной 1 000 000 символов, а также анализируется время, которое тратится задачей на ожидание в очереди.

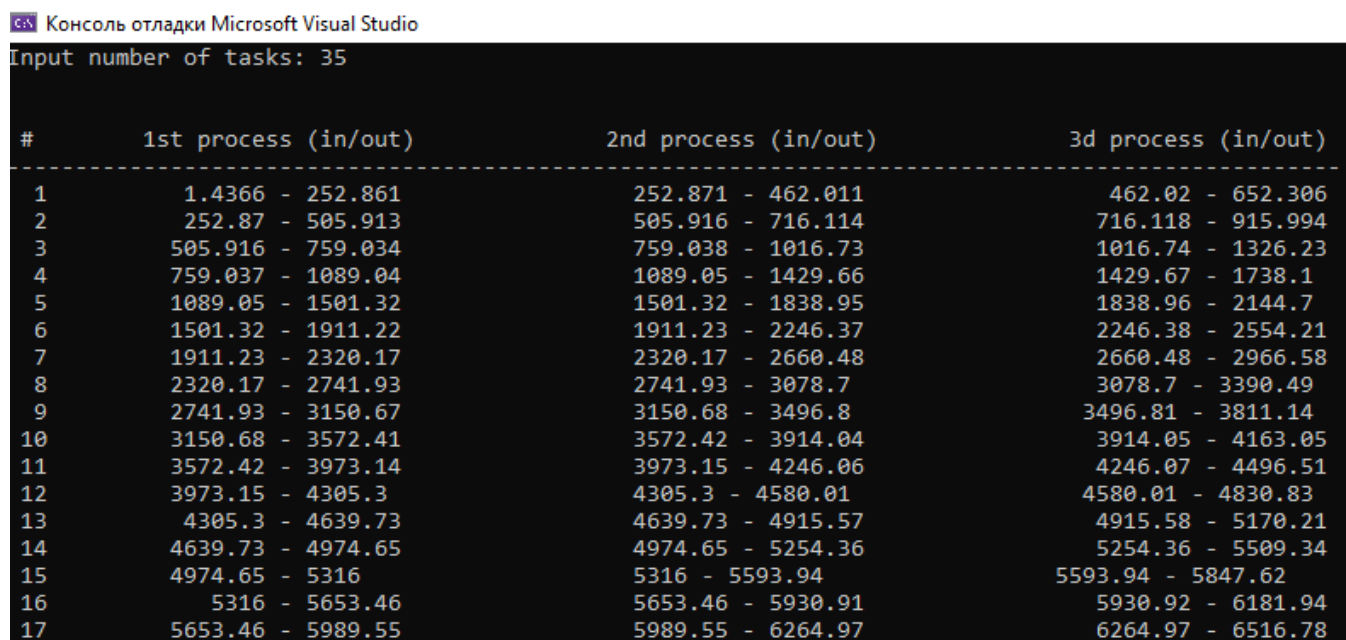
4.1. Характеристики ПК

При проведении эксперимента использовался компьютер, имеющий следующие характеристики:

- ОС - Windows 10 Pro;
- процессор - Intel Core i7 10510U (1800 МГц);
- объём ОЗУ - 16 Гб;
- число логических ядер - 8.

4.2. Лог конвейерной обработки

На Рисунках 4.1, 4.2 приведены результаты обработки 35 задач.



Консоль отладки Microsoft Visual Studio

Input number of tasks: 35

#	1st process (in/out)	2nd process (in/out)	3d process (in/out)
1	1.4366 - 252.861	252.871 - 462.011	462.02 - 652.306
2	252.87 - 505.913	505.916 - 716.114	716.118 - 915.994
3	505.916 - 759.034	759.038 - 1016.73	1016.74 - 1326.23
4	759.037 - 1089.04	1089.05 - 1429.66	1429.67 - 1738.1
5	1089.05 - 1501.32	1501.32 - 1838.95	1838.96 - 2144.7
6	1501.32 - 1911.22	1911.23 - 2246.37	2246.38 - 2554.21
7	1911.23 - 2320.17	2320.17 - 2660.48	2660.48 - 2966.58
8	2320.17 - 2741.93	2741.93 - 3078.7	3078.7 - 3390.49
9	2741.93 - 3150.67	3150.68 - 3496.8	3496.81 - 3811.14
10	3150.68 - 3572.41	3572.42 - 3914.04	3914.05 - 4163.05
11	3572.42 - 3973.14	3973.15 - 4246.06	4246.07 - 4496.51
12	3973.15 - 4305.3	4305.3 - 4580.01	4580.01 - 4830.83
13	4305.3 - 4639.73	4639.73 - 4915.57	4915.58 - 5170.21
14	4639.73 - 4974.65	4974.65 - 5254.36	5254.36 - 5509.34
15	4974.65 - 5316	5316 - 5593.94	5593.94 - 5847.62
16	5316 - 5653.46	5653.46 - 5930.91	5930.92 - 6181.94
17	5653.46 - 5989.55	5989.55 - 6264.97	6264.97 - 6516.78

Рис. 4.1 — Результаты обработки 35 задач (часть 1)

18	5989.55 - 6324.77	6324.77 - 6599.56	6599.56 - 6851.4
19	6324.77 - 6660.04	6660.05 - 6930.63	6930.63 - 7116.82
20	6660.05 - 6974.52	6974.52 - 7179.48	7179.48 - 7368.82
21	6974.52 - 7223.87	7223.88 - 7438.16	7438.17 - 7628
22	7223.88 - 7475.9	7475.9 - 7685.23	7685.24 - 7875.61
23	7475.9 - 7728.32	7728.32 - 7954.28	7954.29 - 8267.23
24	7728.32 - 8025.14	8025.15 - 8366.36	8366.37 - 8681.55
25	8025.15 - 8439.3	8439.3 - 8789.28	8789.28 - 9103
26	8439.3 - 8863.36	8863.37 - 9204.68	9204.68 - 9516.34
27	8863.37 - 9277.14	9277.15 - 9615.1	9615.11 - 9924.87
28	9277.15 - 9687.14	9687.15 - 10026.5	10026.5 - 10336.8
29	9687.15 - 10097.7	10097.7 - 10436.4	10436.4 - 10745.6
30	10097.7 - 10508.5	10508.5 - 10846.7	10846.7 - 11112.7
31	10508.5 - 10919.7	10919.7 - 11195.4	11195.4 - 11448
32	10919.7 - 11254.2	11254.2 - 11567.2	11567.2 - 11826.6
33	11254.2 - 11625.1	11625.1 - 11904.9	11904.9 - 12155.4
34	11625.1 - 11963.3	11963.3 - 12240.3	12240.3 - 12468.3
35	11963.3 - 12298.6	12298.6 - 12543.5	12543.5 - 12729.1

	Min	Max	Avg
Whole task	650.869	1077.85	904.752
Delay	0.0032	0.0239	0.00658429

General time: 12727.7

Рис. 4.2 — Результаты обработки 35 задач (часть 2)

Выводы

Согласно полученным данным, можно сделать следующие **выводы**.

- Среднее время обработки строк длиной 1 000 000 символов примерно 904.752 мс, а среднее время ожидания в очереди 0.00658 мс, что составляет меньше 0.0007% от среднего времени выполнения задачи.
- Если рассматривать наибольшие показатели конвейера, то время ожидания будет составлять примерно 0.0022% от всего времени обработки задачи.
- Соотношение времени ожидания в очереди ко времени выполнения не превышает 0.0037%.

Заключение

Список литературы

1. Конвейерная организация [Электронный ресурс]. Режим доступа: <http://citforum.ru/hardware/appkis/glava34.shtml>, свободный (дата обращения: 01.11.2020)
2. Романьков В.А Введение в криптографию: курс лекций, 2009. — 238 с. — ISBN 5777909825.
3. Фред Б. Риксон. Коды, шифры, сигналы и тайная передача информации. — Астрель, 2011. — ISBN 978-5-17-074391-9.
4. Документация по Стандартной библиотекн языка C++ thread [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/ru-ru/cpp/standard-library/thread?view=vs-2019>, свободный (дата обращения 06.11.2020)
5. Документация по Стандартной библиотекн языка C++ mutex [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/ru-ru/cpp/standard-library/mutex?view=vs-2019>, свободный (дата обращения 06.11.2020)
6. Документация по Стандартной библиотекн языка C++ queue [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/ru-ru/cpp/standard-library/queue-class?view=msvc-160> (дата обращения 06.11.2020)
7. Документация по Visual Studio 2019 [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/ru-ru/visualstudio/windows/?view=vs-2019>, свободный (дата обращения: 05.11.2020)
8. QueryPerformanceCounter function [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancecounter>, свободный (дата обращения: 08.11.2020).