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 **09.03.01 Информатика и вычислительная техника**

О Т Ч Е Т

по лабораторной работе № 5

Название: Разработка программных конвейеров

Дисциплина: Анализ алгоритмов

Студент	ИУ7-52Б		В.А. Иванов
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2020

Оглавление

Введение

В данной лабораторной реализуется и оценивается конвейерный алгоритм шифрования.

Конвейеризация алгоритма — выполнение нескольких последовательных стадий одной задачи в разных потоках. Так же, как и в случае параллелизма, подобное разделение стадий задач между потоками нацелено на получение выигрыша за счёт их параллельного выполнения. Зачастую подобное решение применяется, когда процессоры комьютера оптимизированы под выполнения различного рода операций, что требует распределения стадий задачи.

Алгоритмы шифрования предназначены для защиты данных от прочтения строронними лицами. Существует множество разоичных алгоритмов шифрования, и чтобы обеспечить лучшую защищённость данных иногда прибегают к последовательному шифрованию данных различными алгоритмами. Так как в данной задаче явно прослеживается несколько стадий с сопоставимым временем выполнения, она подходит для реализации программного конвейера.

1. Аналитическая часть

1.1. Цель и задачи работы

Целью лабораторной работы является разработка и исследование конвейерного алгоритма шифования строк.

Выделены следующие задачи лабораторной работы:

- описание понятия конвейеризации и операции шифрования;
- описание и реализация конвейерного алгоритма шифрования строк;
- проведение замеров времени работы и простоя конвейера;

1.2. Описание операции шифрования

Шифрование - операция обратимного изменения вида сообщения в целях сокрытия от неавторизованных лиц. Результатом операции является набор данных отличный от исходного, но имеющий метод преобразования к нему (дешифрование). В данной лабораторной работе исходные и зашифрованные данные представляются в виде символьной строки.

1.3. Используемые алгоритмы

В лабораторной работе был использован шифр, использующий последовательное шифрование тремя алгоритмами[1]. Алгоритмы были выбраны ввиду относительно схожего времени работы на одинаковых строках. Конвейеризация может быть применена ввиду явного разделения шифрования на последовательные этапы. Поэтому конвейер будет обладать тремя лентами, каждая из которых будет выполнять один из алгоритмов шифрования.

1.3.1. Шифр Фейстеля

Основной операцией метода служит XOR. Алгоритм состоит n повторения одной операции. Выбирается два блока данных одинакового размера, после чего производится вычисление значения формулы 1.1

$$(r \oplus ((l + st * 15))\ mod\ N_a, \tag{1.1}$$

где l, r - выбраные блоки данных, st - номер итерации, N_a - размер алфавита строки. После чего данное значение записывается в l, а l, в свою очередь, записывается в r. После чего операция повторяется.

В данной работе в качесте двух блоков данных выступают пары соседних символов строки. Последний символ нечётной строки при таком подходе изменяться не будет.

1.3.2. Шифр Тритемиуса

Данный метод принято называть усложнённым шифром Цезаря. Ключевое отличие заключается в том, что в данном шифре помимо постоянной величины сдвига кодов символов могут использоваться и переменные значения. Например, позиция рассматриваемого символа или функция от этого значения.

В данной работе используется формула осуществления сдвига 1.2

$$str[i] = (pre[i] + floor(\sqrt{i}) - 3 * i + 3)\ mod\ N_a, \tag{1.2}$$

где i - позиция символа в строке, $pre[i]$, $str[i]$ - исходное и зашифрованное значение i-го символа строки, N_a - размер алфавита строки.

1.3.3. Шифр побитовыми сдвигами

Алгоритм использует в себе достаточно быстродейственную операцию циклического побитового сдвига на определённое число позиций. Сначала операция производится для набора байтов от 0-го до k-го, потом от 1-го до (k+1)-го и т.д. до конца строки. Такой метод позволяет достаточно существенно изменить исходный вид строки

1.4. Конвейеризация алгоритма

Идея конвейеризации алгоримта заключается в разделении задачи на стадии и распределение их между лентами конвейера. Каждой ленте выделяется один или более потоков, на которых последовательно отрабатываются одни и те же стадии разных задач. Сами задачи поступают на ленту из очереди для данного конвейера и после обработки лентой переходят в очередь следующей ленты или покидиют конвейер.

В данной реализации каждая лента использует один поток и выполняет шифрование одним из вышеперечисленных алгоритмов. Очередь первой ленты заполняется до начала работы конвейера с помощью генератора случайных строк одинаковой длины. Обработанные задачи поступают в итоговую очередь, где будут обработаны статистические данные времён поступления и уходы с лент конвейера.

Вывод

Результатом аналитического раздела стало определение цели и задач работы, описано понятие опереции шифрования, используемых алгоритмов, конвейеризации. Также были описаны особенности

использования перечисленных методов в данной задаче.

2. Конструкторская часть

Рассмотрим описаные алгоритмы шифрования строки и конвейеризации.

2.1. Алгоритмы шифрования

Рассмотрим работу алгоритов шифрования. Пусть на вход подаётся строка s длины len.

2.1.1. Шифр Фейстеля

Алгоритм циклически обходит пары рядом стоящих символов и производит для них max_step итераций шифрования с помощью операции XOR.

Схема алгоритма приведена на рисунке 2.1

2.1.2. Шифр Тритемиуса

Алгоритм совершает цикл по всем символам строки и для каждого из них совершает вычисление нового значения символа в зависимости от исходнго и текущей позиции

Схема алгоритма приведена на рисунке 2.2

2.1.3. Шифр побитовыми сдвигами

Алгоритм производит циклический сдвиг вправо на $shift$ для buf_byte символов из строки s (с i-й позиции по i+buf_byte), после чего переходит к следующему блоку данных увеличением i на 1.

Схема алгоритма приведена на рисунке 2.3

2.2. Конвейеризация алгоритма

Конвейеризация процесса происходит с помощью алгоритов работы для конвейера (главного потока) и его лент (рабочих пото-

ков).

2.2.1. Алгоритм работы конвейера

Конвейер осуществляет заполнение входной очереди задачами. Задачи являются структурами имеющими в себе 6 полей для записи времени прихода/ухода с лент конвейера и сама задача в виде шифруемой строки. После заполнения, конвейер запускает в работу три ленты конвейера и ожидает их завершения. Дальше конвейер совершает обработку полученной статистики.

Схема алгоритма приведена на рисунке 2.4

2.2.2. Алгоритм работы ленты

Лента производит обработку *queue_len* количества задач. В начале обработки происходит ожидание поступления очередной задачи во входную очередь, после чего она выталкивается из очереди в переменную задачи. Выполняется функция шифрования, записывается время начала и конца обработки. Обработанная структура записывается в выходную очередь

Схема алгоритма приведена на рисунке 2.5

2.3. Требования к программному обеспечению

Для полноценной проверки и оценки алгоритмов необходимо выполнить следующее.

1. Предоставить возможность ввода количества задач.
2. Реализовать функцую вывода информации о каждой обработанной задаче и функции подсчёта и вывода статистических данных.

3. Реализовать функцию замера процессорного времени, затраченного функциями.

2.4. Заготовки тестов

При проверке алгоритма необходимо будет использовать следующие классы тестов:

- одна задача;
- 1000 задач.

Вывод

Результатом конструторской части стало схематическое описание алгоритмов шифрования строк и конвейеризации алгоритма, сформулированны тесты и требования к программному обеспечению.

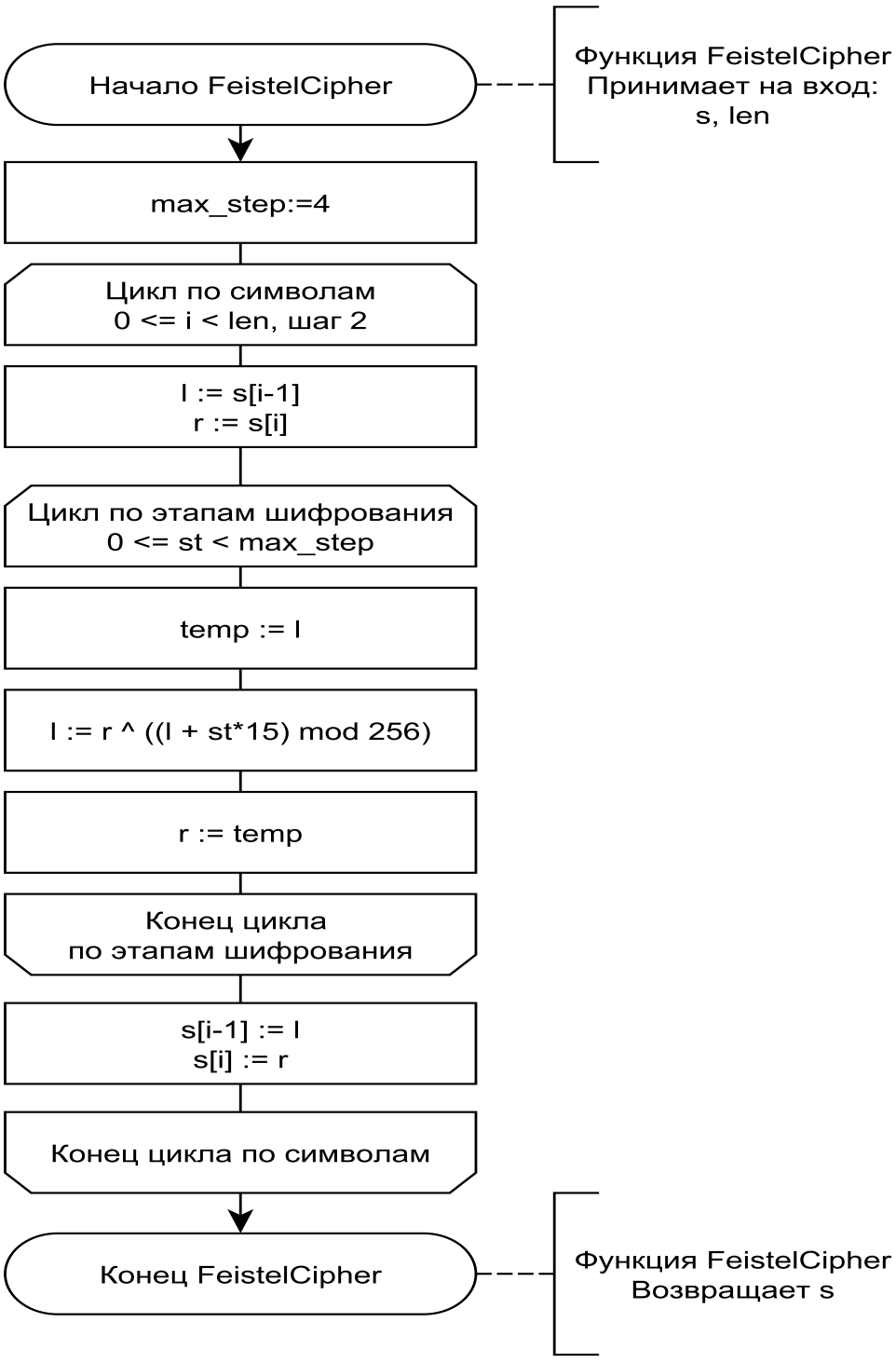

Рис. 2.1 — Шифр Фейстеля

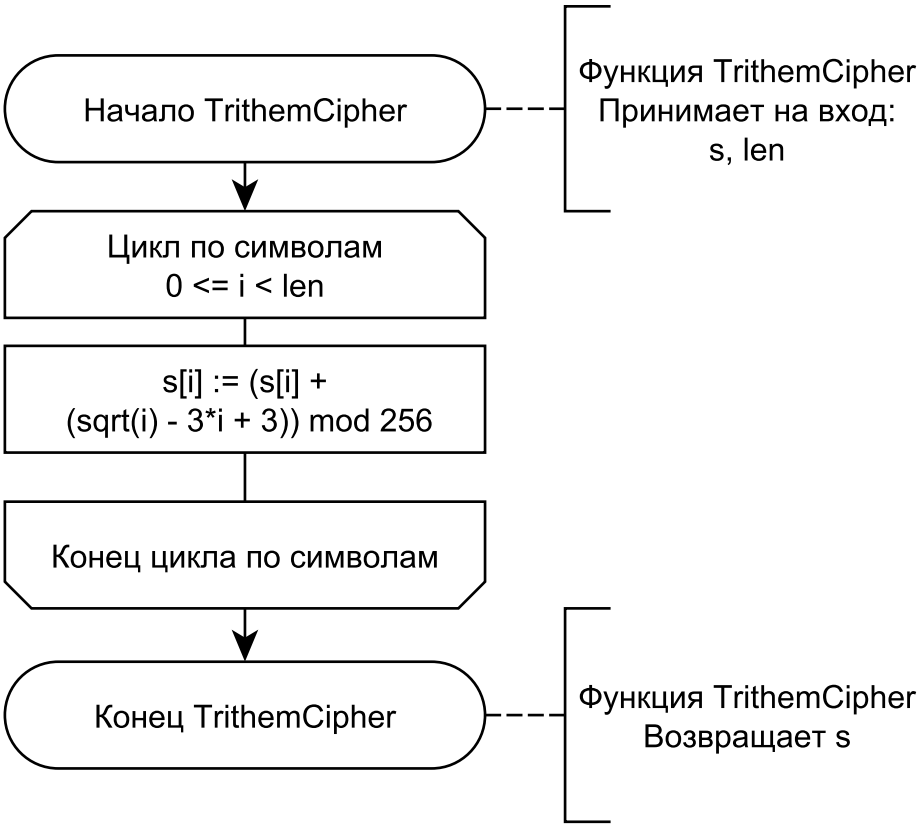

Рис. 2.2 — Шифр Тритемиуса

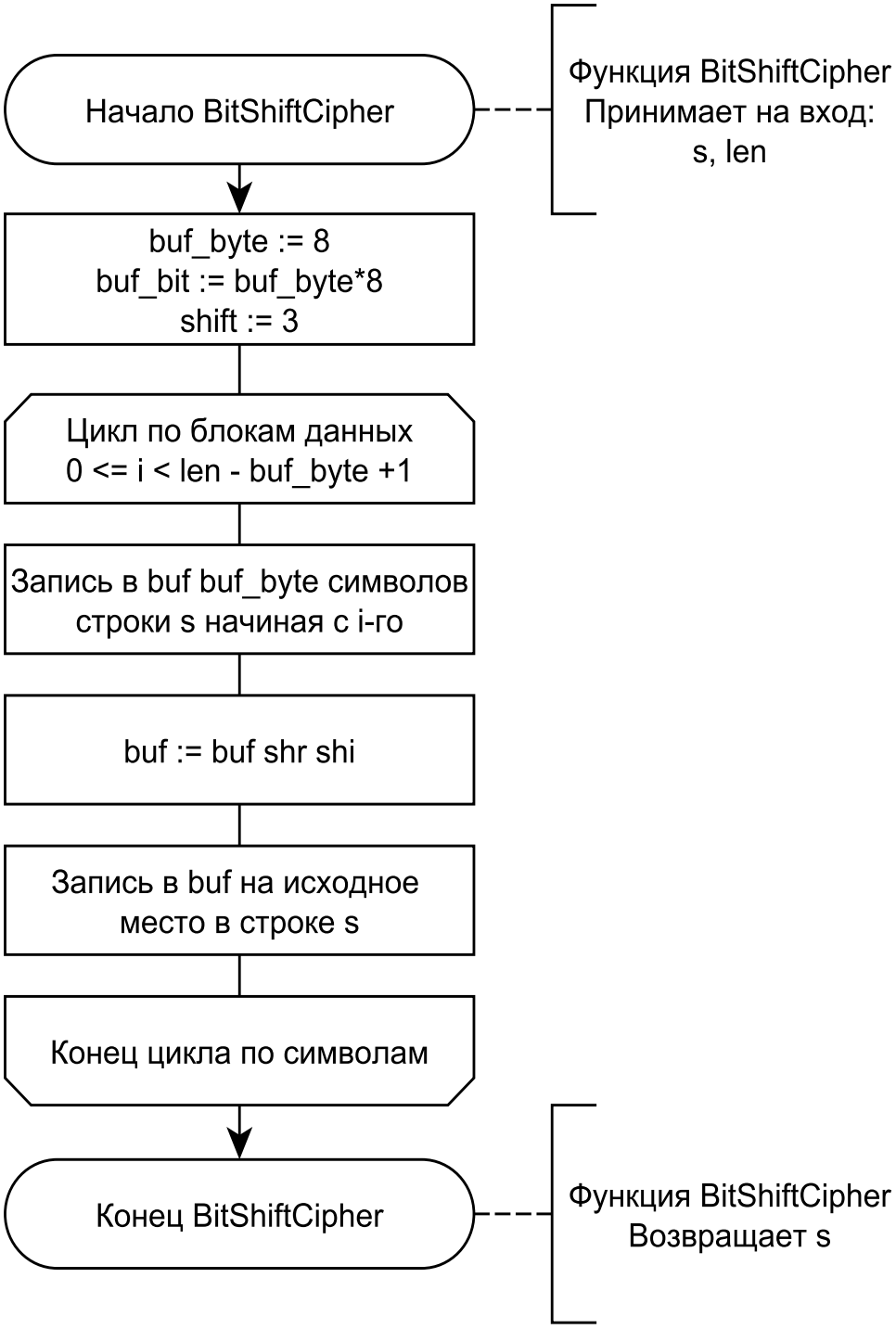

Рис. 2.3 — Шифр побитовыми сдвигами

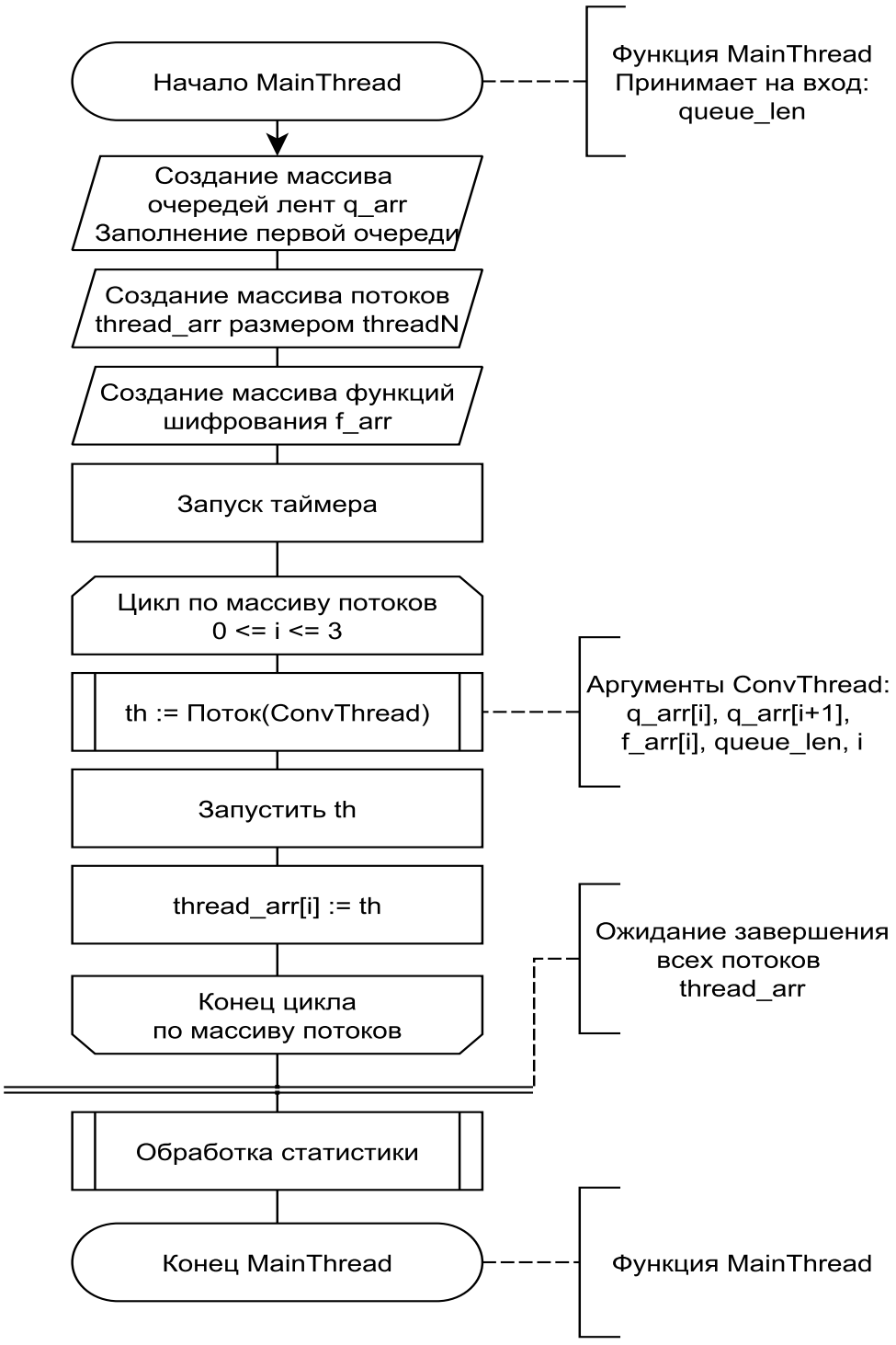

Рис. 2.4 — Алгоритм работы конвейера

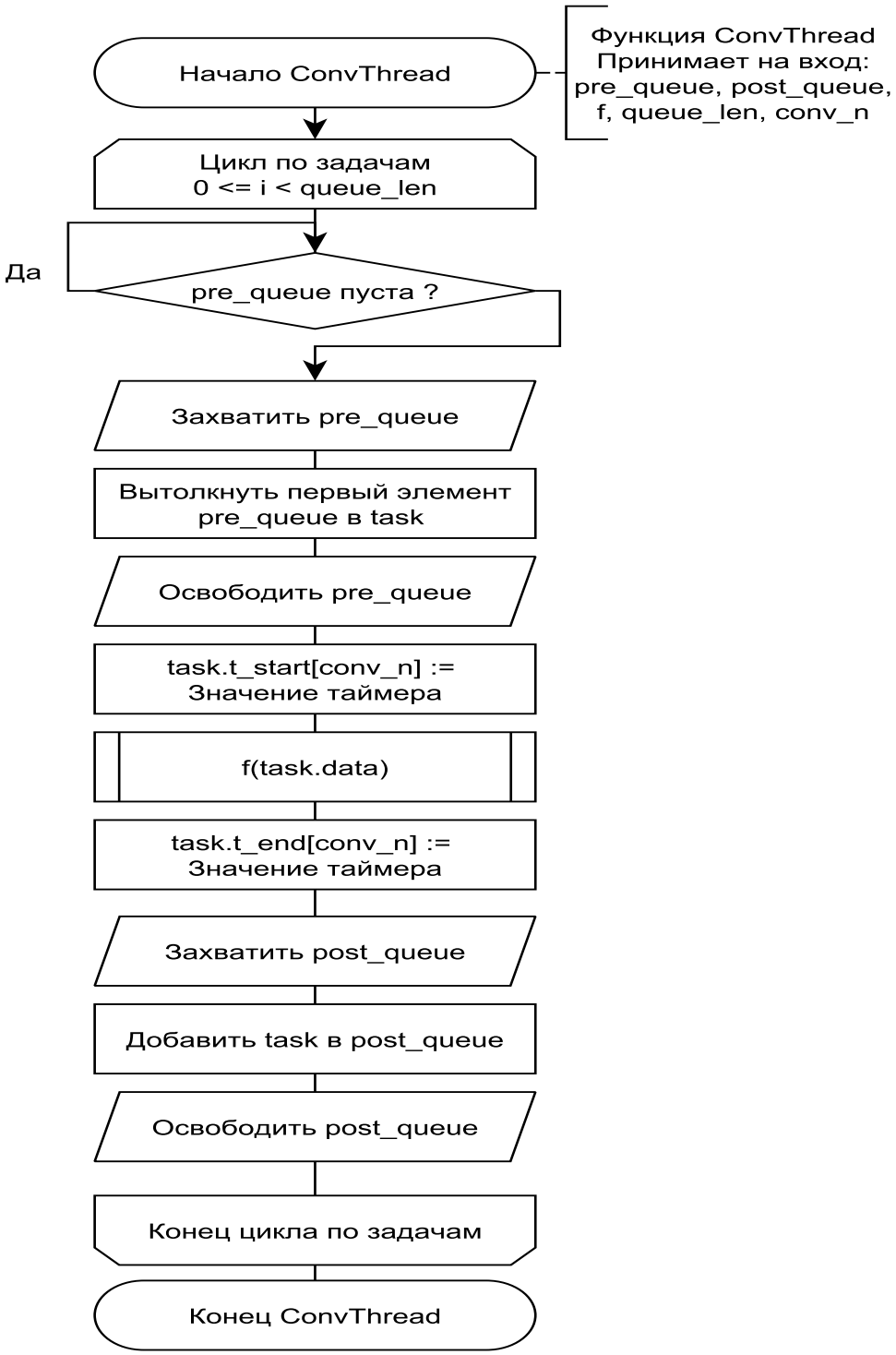

Рис. 2.5 — Алгоритм работы ленты

3. Технологическая часть

3.1. Выбор языка программирования

В качестве языка программирования был выбран C++[2], так как имеется опыт работы с ним, и с библиотеками, позволяющими провести исследование и тестирование программы. Также в языке имеются средства для использования многопоточности[3]. Разработка проводилась в среде Visual Studio 2019[4].

3.2. Листинги кода

Реализация алгоритмов шифрования строк представлена на листингах 3.1-3.3. Функции конвейера и ленты приведены на листингах 3.4-3.5

Листинг 3.1 — Функция шифра Фейстеля

```
1  void FeistelCipher(string& s)
2  {
3    const int max_step = 4;
4    int l, r, temp;
5
6    for (int i = 1; i < s.length(); i += 2)
7    {
8      l = s[i - 1];
9      r = s[i];
10
11     for (int st = 0; st < max_step; st++)
12     {
13       temp = l;
14       l = r ^ ((l + st * 15) % 256);
15       r = temp;
16     }
17     s[i - 1] = l;
18     s[i] = r;
19   }
```

```
20 }
```

Листинг 3.2 — Функция шифра Тритемиуса

```
1 void TrithemiusCipher(string& s)
2 {
3   for (int i = 0; i < s.length(); i++)
4   {
5     s[i] = (s[i] + (int(sqrt(i)) - 3 * i + 3)) % 256;
6   }
7 }
```

Листинг 3.3 — Функция шифра побитовыми сдвигами

```
1 using buf_t = long unsigned int;
2 void BitShiftCipher(string& s)
3 {
4   buf_t buf;
5   int buf_byte = sizeof(buf_t);
6   int buf_bit = buf_byte * 8;
7   int shift = 3;
8
9   for (int i = 0; i < s.size() - buf_byte + 1; i++)
10  {
11    memcpy(&buf, &s[i], buf_byte);
12    buf = (buf >> shift) | (buf << (buf_bit - shift));
13    memcpy(&s[i], &buf, buf_byte);
14  }
15 }
```

Листинг 3.4 — Функция основного потока (конвейера)

```
1 void process_data(conv_queue& task_queue)
2 {
3   int queue_len = task_queue.size();
4   conv_shared task;
```

```
5
6   double t_wait, t_proc;
7   double min_wait, max_wait, sum_wait;
8   double min_proc, max_proc, sum_proc;
9   min_wait = 1e10;     min_proc = 1e10;
10  max_wait = -1;       max_proc = -1;
11  sum_wait = 0;        sum_proc = 0;
12  for (int i = 0; i < queue_len; i++)
13  {
14    task = task_queue.front();
15    task_queue.pop();
16    t_wait = (task->t_start[1] - task->t_end[0]) +
17    (task->t_start[2] - task->t_end[1]);
18    min_wait = min(min_wait, t_wait);
19    max_wait = max(max_wait, t_wait);
20    sum_wait += t_wait;
21
22    t_proc = task->t_end[2] - task->t_start[0];
23    min_proc = min(min_proc, t_proc);
24    max_proc = max(max_proc, t_proc);
25    sum_proc += t_proc;
26  }
27
28  cout << endl << endl;
29  printf("WAITING TIME:\t Min:%6.2lf\t Max:%6.2lf\t Avg:%6.2
     lf\n",
30  min_wait, max_wait, sum_wait / queue_len);
31  printf("PROCESS TIME:\t Min:%6.2lf\t Max:%6.2lf\t Avg:%6.2
     lf\n",
32  min_proc, max_proc, sum_proc / queue_len);
33 }
34
35 void main_thread(int queue_len)
36 {
```

```
37    fCipher f_arr[] = { FeistelCipher, TrithemiusCipher,
       BitShiftCipher };
38    vector<thread> thread_arr;
39    conv_queue q_arr[4];
40    q_arr[0] = generate_queue(queue_len, 1000*1000);
41
42    start_counter();
43    for (int i = 0; i < 3; i++)
44    thread_arr.push_back(thread(conv_thread,
45    ref(q_arr[i]), ref(q_arr[i+1]), f_arr[i], queue_len, i));
46
47    for (int i = 0; i < 3; i++)
48    thread_arr[i].join();
49
50    process_data(q_arr[3]);
51  }
```

Листинг 3.5 — Функция рабочего потока (ленты)

```
1  void conv_thread(conv_queue& pre_queue, conv_queue&
       post_queue, fCipher f, int queue_len, int conv_n)
2  {
3    conv_shared task;
4    for (int i = 0; i < queue_len; i++)
5    {
6      while (pre_queue.empty()) {}
7      task = pre_queue.front();
8      task->t_start[conv_n] = get_counter();
9      if (conv_n == 0)
10     {
11       pre_queue.pop();
12     }
13     else
14     {
```

```
15          mtx.lock();
16          pre_queue.pop();
17          mtx.unlock();
18      }
19
20      f(task->data);
21
22      task->t_end[conv_n] = get_counter();
23      if (conv_n == 2)
24      {
25          post_queue.push(task);
26      }
27      else
28      {
29          mtx.lock();
30          post_queue.push(task);
31          mtx.unlock();
32      }
33  }
34 }
```

3.3. Результаты тестирования

Ввиду крайне ограниченой зависимости от входных данных, тестирование проводилось вручную путём введения вышевописанных случаев через консоль. Все тесты пройдены успешно.

3.4. Оценка времени

Для замера процессорного времени исполнения функции используется функция QueryPerformanceCounter библиотеки windows.h[5]. Код функций замера времени приведёны в листинге 3.6.

Листинг 3.6 — Функции замера процессорного времени работы функции

```
1
2 double PCFreq = 0.0;
3 __int64 CounterStart = 0;
4
5 void start_counter()
6 {
7   LARGE_INTEGER li;
8   QueryPerformanceFrequency(&li);
9
10  PCFreq = double(li.QuadPart) / 1000.0;
11
12  QueryPerformanceCounter(&li);
13  CounterStart = li.QuadPart;
14 }
15
16 double get_counter()
17 {
18  LARGE_INTEGER li;
19  QueryPerformanceCounter(&li);
20  return double(li.QuadPart - CounterStart) / PCFreq;
21 }
```

Вывод

Результатом технологической части стал выбор используемых технических средств реализации и реализация алгоритмов, системы тестов и замера времени работы на языке C++.

4. Исследовательская часть

4.1. Описание эксперимента

Эксперимент работы конвейера проводится на строчках длиной 2 миллиона символов, всего обработано 50 задач. В качестве результата приведены данные о времени начала и завершения обработки конвеерами для каждой задачи. Также вычисляются и демонстрируются минимально, максимальное и среднее время обработки задачи и простоя в очереди.

4.2. Результат эксперимента

Результаты эксперимента представлены на рисунке 4.1. Время указано в милисекундах.

4.3. Характеристики ПК

Эксперименты проводились на компьютере с характеристиками:

- ОС - Windows 10, 64 бит;
- Процессор - Intel Core i7 8550U (1800 МГц, 4 ядра, 8 логических процессоров);
- Объем ОЗУ: 8 ГБ.

Вывод

По результатам экспериментов можно заключить следующее.

- Среднее время обработки задачи примерно в 160 раз больше времени ожидания задачи в очереди и диспетчеризации.
- Максимальное время простоя в очереди приблезительно в 5 раз меньше среднего времени обработки задчи.

- Оба предыдущих вывода показывают, что время диспетчеризации в любом случае и простоя значительно меньше, чем время активной обработки задачи лентами конвейера.
- Минимальное и максимальное время обработки имеют различие лишь на 30
- Ввиду пренебрежимости времени диспетчеризации, можно утверждать, что минимальное время обработки задачи равно времени, которое было бы затрачено в случае использования неконвейерного алгоритма. Тогда в среднем каждая задача в конвейерном алгоритме обрабатывается лишь на 6% дольше, чем в алгоритме без конвейера.
- На обработку всех 50 задач было затрачено примерно 660 мс. Из пункта выше можно предположить, что алгоритм без конвейера затратил был $30.5 * 50 = 1525$ мс, что в 2.3 раза дольше, чем в изученной реализации.

```
Введите количество задач: 50
  0|       1,52:  13,89          13,89:  23,68          23,68:  32,76
  1|      13,89:  25,99          25,99:  36,21          36,21:  45,41
  2|      25,99:  38,76          38,77:  49,16          49,17:  58,42
  3|      38,77:  51,65          51,65:  62,04          62,04:  70,88
  4|      51,65:  64,51          64,52:  74,36          74,36:  83,20
  5|      64,52:  76,74          76,74:  86,51          86,51:  95,53
  6|      76,74:  88,90          88,90:  98,53          98,53: 107,35
  7|      88,90: 102,35         102,35: 112,14         112,15: 121,02
  8|     102,35: 114,50         114,50: 126,67         126,67: 135,42
  9|     114,50: 128,47         128,47: 138,48         138,48: 147,29
 10|     128,47: 140,72         140,72: 150,57         150,57: 159,43
 11|     140,72: 152,85         152,85: 171,02         171,02: 180,10
 12|     152,85: 164,96         171,02: 180,52         180,52: 189,53
 13|     164,97: 179,24         180,52: 190,48         190,48: 199,38
 14|     179,24: 193,63         193,63: 203,64         203,64: 212,47
 15|     193,63: 205,79         205,79: 215,74         215,74: 224,55
 16|     205,79: 217,98         217,98: 227,76         227,76: 236,29
 17|     217,98: 230,23         230,23: 245,54         245,54: 254,02
 18|     230,23: 245,57         245,57: 258,91         258,91: 267,75
 19|     245,57: 259,88         259,88: 269,80         269,80: 278,67
 20|     259,88: 272,06         272,06: 281,88         281,88: 290,68
 21|     272,06: 284,22         284,22: 294,02         294,02: 302,90
 22|     284,22: 296,39         296,39: 306,31         306,31: 315,20
 23|     296,39: 308,58         308,58: 318,45         318,45: 327,28
 24|     308,58: 320,85         320,85: 330,69         330,70: 339,50
 25|     320,85: 333,36         333,36: 343,13         343,13: 352,03
 26|     333,36: 345,71         345,71: 355,50         355,50: 364,15
 27|     345,71: 358,10         358,11: 373,30         373,30: 381,87
 28|     358,11: 373,57         373,57: 386,84         386,84: 395,72
 29|     373,57: 387,89         387,89: 397,71         397,71: 406,55
 30|     387,89: 400,01         400,01: 409,92         409,92: 418,74
 31|     400,01: 412,18         412,18: 421,99         421,99: 430,81
 32|     412,19: 424,35         424,35: 434,12         434,12: 442,94
 33|     424,35: 436,48         436,48: 446,31         446,31: 455,21
 34|     436,48: 448,71         448,71: 463,28         463,28: 472,11
 35|     448,71: 460,96         463,28: 473,09         473,09: 481,90
 36|     460,96: 473,17         473,17: 482,95         482,95: 491,79
 37|     473,17: 485,38         485,38: 495,15         495,15: 503,98
 38|     485,38: 497,55         497,55: 507,36         507,36: 516,19
 39|     497,55: 509,71         509,71: 519,51         519,51: 528,52
 40|     509,71: 521,93         521,94: 531,70         531,70: 540,62
 41|     521,94: 534,15         534,15: 543,94         543,94: 552,78
 42|     534,15: 546,32         546,32: 556,12         556,12: 564,82
 43|     546,33: 558,52         558,52: 571,60         571,60: 580,23
 44|     558,52: 572,76         572,76: 583,21         583,21: 592,03
 45|     572,76: 585,37         585,37: 595,10         595,10: 604,04
 46|     585,37: 598,60         598,60: 607,99         607,99: 617,03
 47|     598,60: 613,24         613,24: 622,58         622,58: 631,70
 48|     613,24: 628,28         628,28: 637,70         637,70: 646,39
 49|     628,28: 642,60         642,60: 652,00         652,00: 660,57


WAITING TIME:    Min:  0,00     Max:  6,05     Avg:  0,19
PROCESS TIME:    Min: 30,50     Max: 39,37     Avg: 32,42
```

Рис. 4.1 — Результаты эксперимента

Заключение

В ходе лабораторной работы достигнута поставленная цель: разработка и исследование конвейерного алгоритма шифования строк. Решены все задачи работы.

Были изучены и описаны понятия конвейеризации и операции шифрования. Также были описан и реализован конвейерного алгоритма шифрования строк. Проведены замеры процессорного времени работы и простоя конвейера. На основании экспериментов проведён сравнительный анализ.

Из проведённых экспериментов было выявлено, что при использовании конвейерного алгоритма можно получить время обработки одной задачи сравнимое с непараллельным вариантом. Это выражено в результатах оценочного сравнения времени ожидания и общего времени обработки, а также сравнения с минимальными и максимальными величинами обработок и простоев. Значительного отличия минимального и максимального времени обработки отличаются незначительно, из чего можно сделать вывод о приемлемом размере очередей в момент пиковой загрузки.

Список литературы

1. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. М.: Триумф, 2003. 806 с.

2. Документация языка C++ 98 [Электронный ресурс]. Режим доступа: http://www.open-std.org/JTC1/SC22/WG21/, свободный (дата обращения: 14.10.2020)

3. Документация библиотеки std (раздел о потоках) [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/thread, свободный (дата обращения: 23.10.2020).

4. Документация среды разработки Visual Studio 2019 [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/visualstudio/windows/?view=vs-2019, свободный (дата обращения: 14.10.2020)

5. QueryPerformanceCounter function [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancecounter, свободный (дата обращения: 29.09.2020).