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: «Конвейерная обработка данных»

Студент: Пронин А. С.

Группа: ИУ7-52Б

Преподаватель: Волкова Л. Л.

Оценка: _____

Москва

2021

Содержание

Введение

Цель работы – получить навык организации асинхронной передачи данных между потоками на примере конвейерной обработки информации.

Задачи работы:

- выбрать и описать методы обработки данных, которые будут сопоставлены методам конвейера;
- реализовать конвейерную систему, а также сформировать лог событий с указанием времени их происхождения;
- сравнить реализации с конвейером и без.

1 Аналитический раздел

Выполнение каждой команды складывается из ряда последовательных этапов (шагов, стадий), суть которых не меняется от команды к команде. С целью увеличения быстродействия процессора и максимального использования всех его возможностей в современных микропроцессорах используется конвейерный принцип обработки информации. Этот принцип подразумевает, что в каждый момент времени процессор работает над различными стадиями выполнения нескольких команд, причем на выполнение каждой стадии выделяются отдельные аппаратные ресурсы. По очередному тактовому импульсу каждая команда в конвейере продвигается на следующую стадию обработки, выполненная команда покидает конвейер, а новая поступает в него.

Конвейерная обработка в общем случае основана на разделении подлежащей исполнению функции на более мелкие части и выделении для каждой из них отдельного блока аппаратуры. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд. Конвейерная обработка такого рода широко применяется во всех современных быстродействующих процессорах.

1.1 Решаемая конвейером задача

В данной лабораторной работе выбран следующий алгоритм для реализации: поиск наибольшего полиндрома в строке. Данный алгоритм можно разбить на 3 этапа:

- разбить строку на слова;
- найти все полиндромы в полученных словах;
- найти наибольший полиндром из всех;

1.2 Вывод

В данном разделе была расмотрена концепция конвейера и выбран алгоритм для реализации.

2 Конструкторский раздел

В данном разделе представлены схемы алгоритмов работы главного потока и конвейера.

2.1 Схемы алгоритмов

Схема алгоритма работы главного потока представлена на рисунке 2.1. Схема алгоритма работы ленты конвейера представлена на рисунке 2.2. Здесь представлен общий вид схемы, а под обработкой заявки понимаются разные алгоритмы. Лента 1 выполняет алгоритм, представленный на рис. 2.3, лента 2 – алгоритм, представленный на рис. 2.4

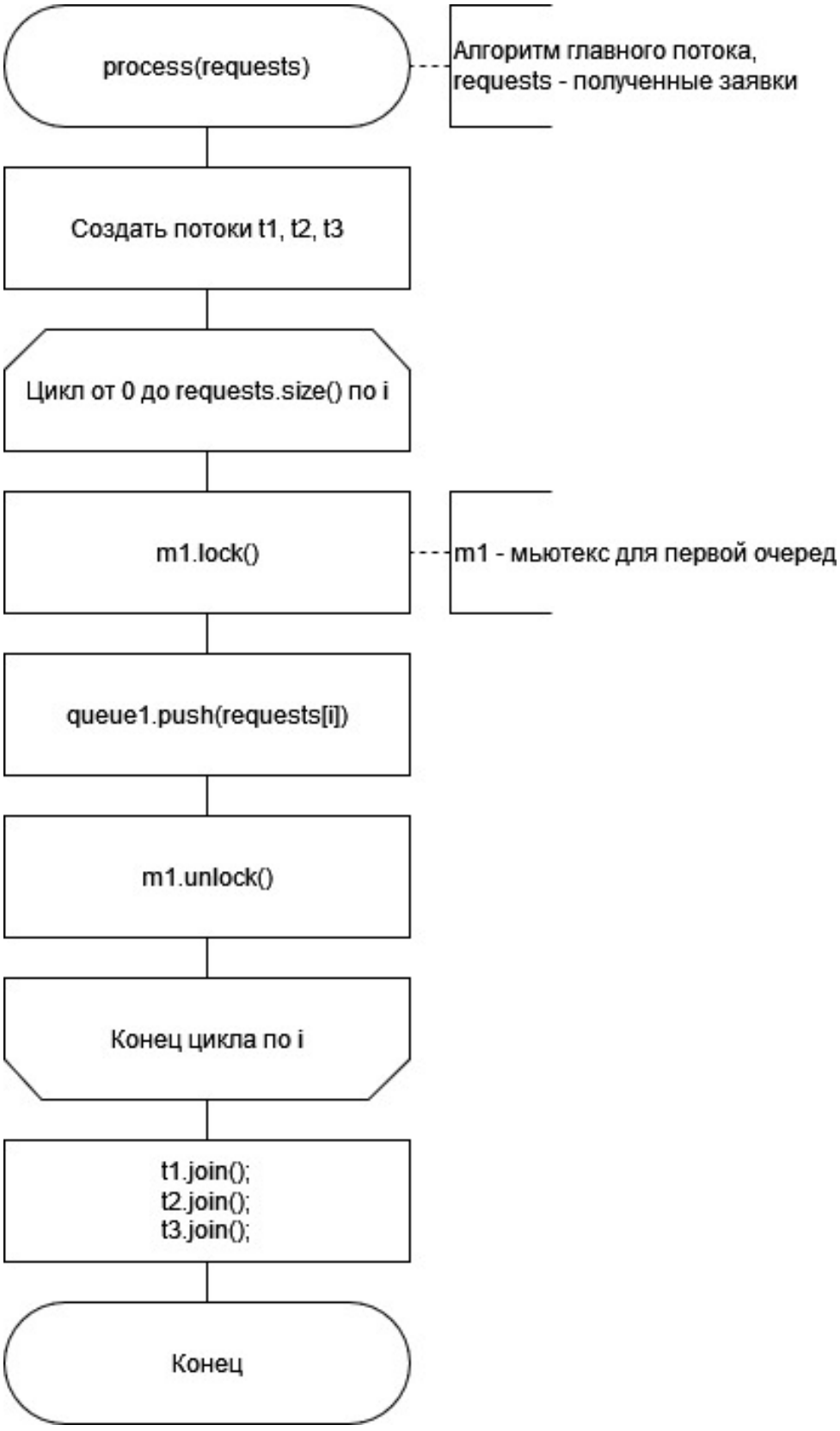

Рис. 2.1: Схема алгоритма работы главного потока

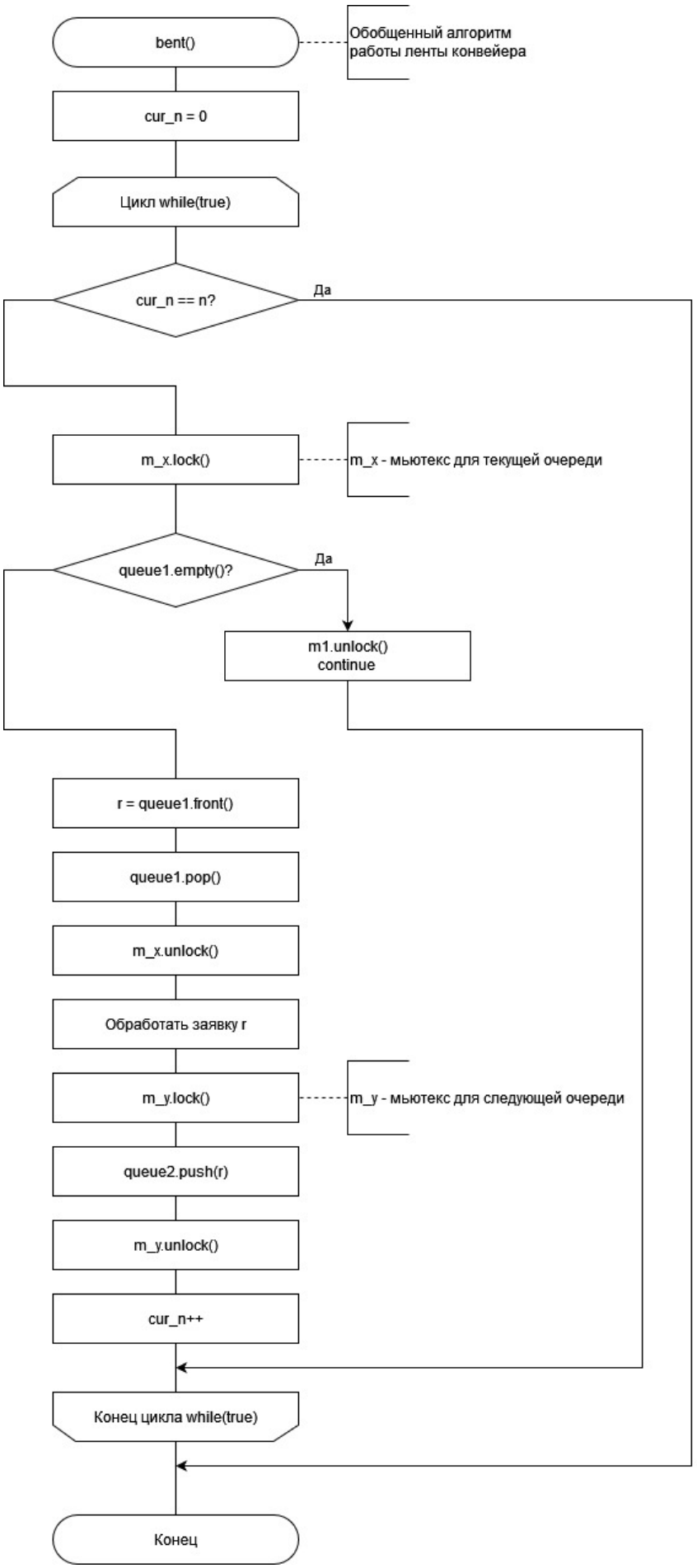

Рис. 2.2: Обобщенная схема алгоритма работы ленты конвейера

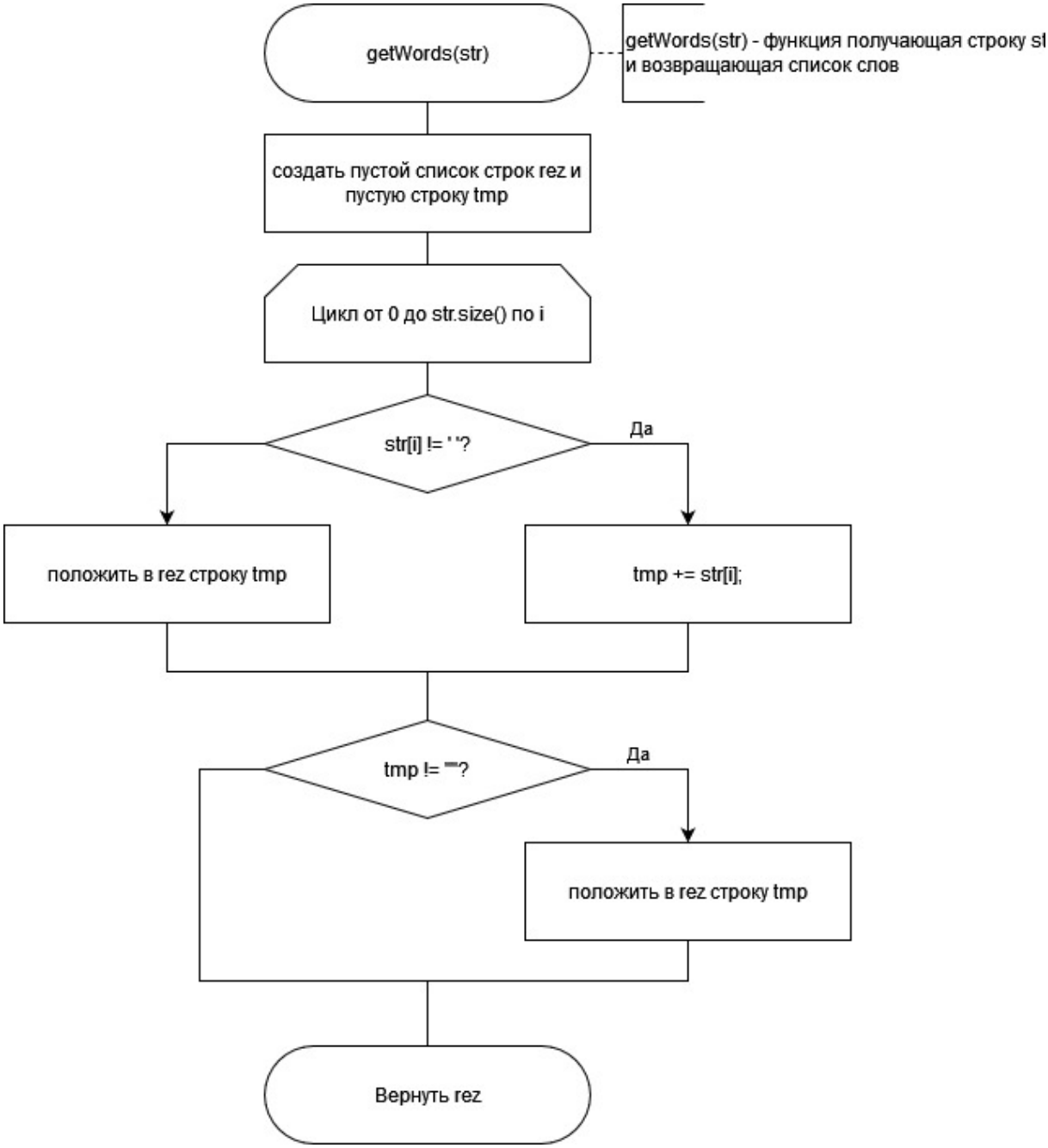

Рис. 2.3: Схема алгоритма разбиения строки на слова

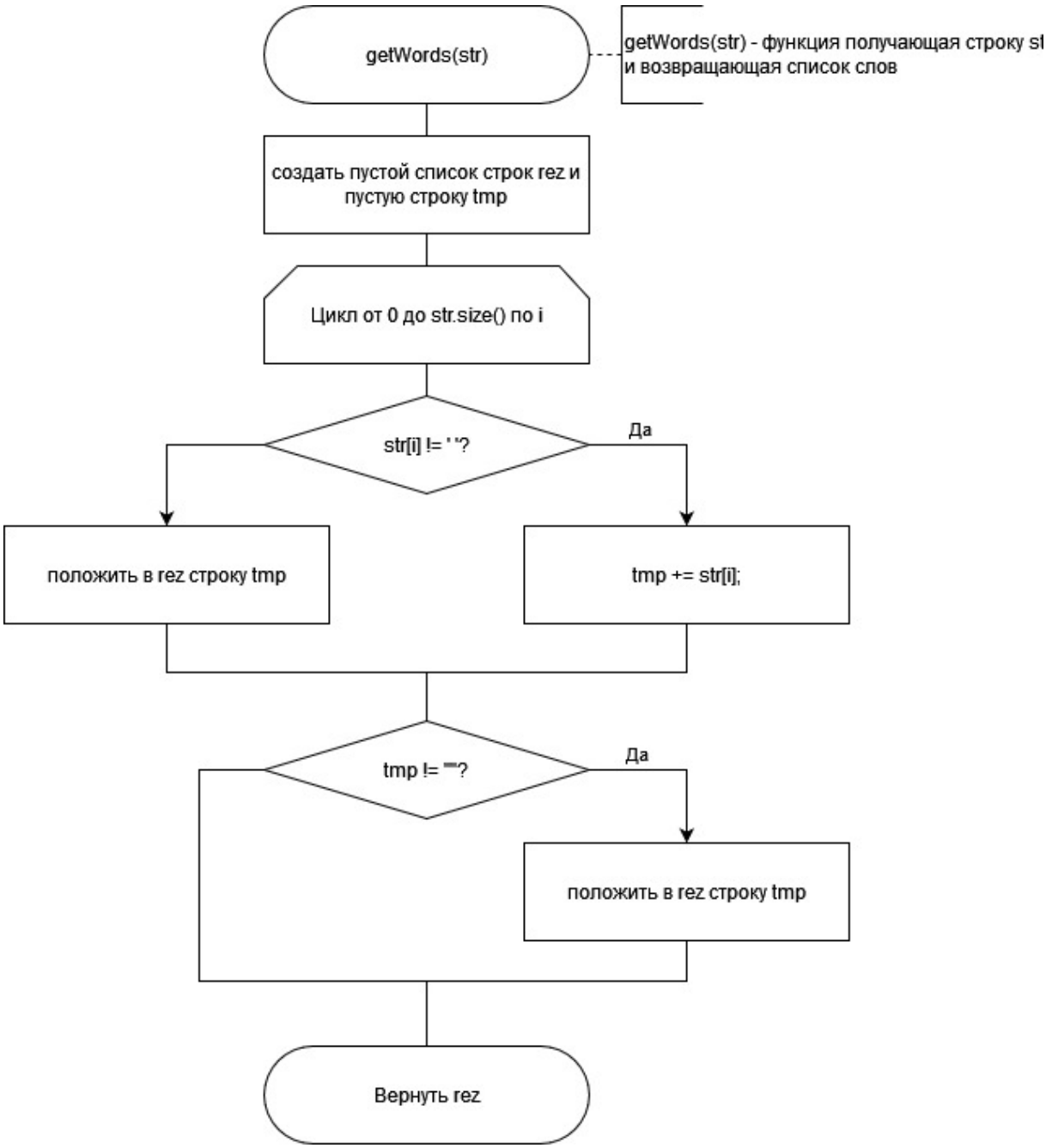

Рис. 2.4: Схема алгоритма нахождения всех полиндромов из полученных слов

2.2 Вывод

В данном разделе были разработаны схемы алгоритмов работы главного потока, обобщенная схема алгоритма работы ленты конвейера и схемы алгоритмов обработки заявок.

3 Технологический раздел

В данном разеделе представлены требование к ПО, выбор инструментов для реализации и оценки алгоритмов, а также листинги полученного кода.

3.1 Требование к ПО

К программе предъявляется ряд требований:

- на вход подаётся строка;
- на выходе — самый длинный полиндром, если таких несколько, то первый в строке.

3.2 Выбор инструментов

Поскольку наиболее освоенным языком для разработчика является c++, для реализации поставленной задачи был выбран именно он, т.к. таким образом работа будет проделана наиболее быстро и качественно.

Соответственно для компиляции кода будет использоваться компилятор G++.

Чтобы оценить время выполнения программы, будет замерятся реальное время, т.к. таким образом можно будет сравнить реализацию с конвейером и без. Для замера реального времени работы программы используется функция $chrono :: high\_resolution\_clock :: now()$ т.к. программа апробируется на компьютере с установленной ОС Windows. [1]

Кроме этого, необходимо отключить оптимизации компилятора для более честного сравнения алгоритмов. В моём случае это делается с помощью ключа $-O0$ т.к. используется компилятор G++. [2]

3.3 Реализация алгоритмов

На листингах 3.1-3.4 представлены реализации алгоритма работы главного потока и реализации алгоритмов лент конвейера.

Листинг 3.1: Реализация алгоритма работы главного потока

```
void Conveyor::process(vector<Request> &requests)
{
    start_time = Clock::now();

    thread t1(&Conveyor::firstBent, this);
    thread t2(&Conveyor::secondBent, this);
    thread t3(&Conveyor::thirdBent, this);

    for (size_t i = 0; i < requests.size(); i++)
    {
        m1.lock();
        requests[i].push_time1 = duration_cast<nanoseconds>(Clock::now() -
            start_time).count()/1000000000.;
        queue1.push(requests[i]);
        m1.unlock();
        Sleep(10);
    }

    t1.join();
    t2.join();
    t3.join();
}
```

Листинг 3.2: Реализация алгоритма первой ленты конвейера

```
void Conveyor::firstBent()
{
    int cur_n = 0;
    while (true)
    {
        if (cur_n == n)
            break;
        m1.lock();
        if (queue1.empty())
        {
            m1.unlock();
            continue;
        }
        Request r = queue1.front();
        queue1.pop();
        r.pop_time1 = duration_cast<nanoseconds>(Clock::now() -
            start_time).count()/1000000000.;
        m1.unlock();
        r.getWords();
        m2.lock();
        r.push_time2 = duration_cast<nanoseconds>(Clock::now() -
            start_time).count()/1000000000.;
        queue2.push(r);
        m2.unlock();
        cur_n ++;
    }
}
```

Листинг 3.3: Реализация алгоритма второй ленты конвейера

```
void Conveyor::secondBent()
{
    int cur_n = 0;
    while (true)
    {
        if (cur_n == n)
            break;
        m2.lock();
        if (queue2.empty())
        {
            m2.unlock();
            continue;
        }
        Request r = queue2.front();
        queue2.pop();
        r.pop_time2 = duration_cast<nanoseconds>(Clock::now() -
            start_time).count()/1000000000.;
        m2.unlock();
        r.getPolinoms();
        m3.lock();
        r.push_time3 = duration_cast<nanoseconds>(Clock::now() -
            start_time).count()/1000000000.;
        queue3.push(r);
        m3.unlock();
        cur_n ++;
    }
}
```

Листинг 3.4: Реализация алгоритма третьей ленты конвейера

```
void Conveyor::thirdBent()
{
    int cur_n = 0;
    while (true)
    {
        if (cur_n == n)
            break;
        m3.lock();
        if (queue3.empty())
        {
            m3.unlock();
            continue;
        }
        Request r = queue3.front();
        queue3.pop();
        r.pop_time3 = duration_cast<nanoseconds>(Clock::now() -
            start_time).count()/1000000000.;
        m3.unlock();
        r.getLongestPolinom();
        r.processing_time = duration_cast<nanoseconds>(Clock::now() -
            start_time).count()/1000000000.;

        m4.lock();
        res.push_back(r);
        m4.unlock();
        cur_n ++;

    }
}
```

3.4 Тестирование

Для проверки написанных алгоритмов были подготовлены следующие тесты:

- проверка реузльтата обработки строки "test lol 23323232 s sss ll 2332323322 sls sll lls"
- проверка реузльтата обработки строки "test lol 23323232 s sss ll 233232332 sls sll lls"
- проверка реузльтата обработки строки "te str"

Для подготовленных тестов ожидаются следующие результаты соответсвтенно:

- "lol"
- "233232332"
- "No polinoms"

На рисунке 3.1 приведены результаты тестирования. Тестирование проводилось по методу чёрного ящика.

```
Request #1:
str = test lol 23323232 s sss ll 2332323322 sls sll lls
words = test; lol; 23323232; s; sss; ll; 2332323322; sls; sll; lls;
polinoms = lol; s; sss; ll; sls;
longest_polinom = lol

Request #2:
str = test lol 23323232 s sss ll 233232332 sls sll lls
words = test; lol; 23323232; s; sss; ll; 233232332; sls; sll; lls;
polinoms = lol; s; sss; ll; 233232332; sls;
longest_polinom = 233232332

Request #3:
str = te str
words = te; str;
polinoms =
longest_polinom = No polinoms
```

Рис. 3.1: Результаты тестирования

Как видно, все тесты пройдены.

3.5 Вывод

В данном разделе были выдвинуты требования к ПО, выбраны инструменты реализации выбранных алгоритмов, представлены листинги реализованных алгоритмов, а также проведено тестирование.

4 Исследовательский раздел

В данном разделе представлены технические характеристики компьютера, используемого для тестирования и экспериментов, и результат работы программы.

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором были проведены эксперименты при помощи разработанного ПО:

- операционная система: Windows 10 (64-разрядная);
- оперативная память: 32 GB;
- процессор: Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz;
- количество ядер: 4;
- количество потоков: 8.

4.2 Результат рыботы программы

На рисунке 4.1 представлены результаты работы программы для обработки 10 заявок, каждая из которых содержит строку из 10000 слов, состоящих из 1-10 букв.

```
Conveyor time:
194276500 ns
194.276 ms
0.194276 s

|Reqest # | push_time1 |  pop_time1 | push_time2 |  pop_time2 | push_time3 |  pop_time3 |  proc_time |
|---------|------------|------------|------------|------------|------------|------------|------------|
|       1 |   0.002001 |   0.002001 |   0.008001 |   0.009003 |   0.020000 |   0.022000 |   0.036000 |
|       2 |   0.020000 |   0.020000 |   0.042000 |   0.043001 |   0.052000 |   0.054001 |   0.066691 |
|       3 |   0.036000 |   0.043001 |   0.056000 |   0.058002 |   0.066691 |   0.068693 |   0.082699 |
|       4 |   0.052000 |   0.057003 |   0.070692 |   0.071690 |   0.082699 |   0.084703 |   0.098678 |
|       5 |   0.066691 |   0.071690 |   0.088702 |   0.090703 |   0.098678 |   0.100678 |   0.114688 |
|       6 |   0.082699 |   0.089704 |   0.102678 |   0.103678 |   0.114688 |   0.116689 |   0.130705 |
|       7 |   0.098678 |   0.103678 |   0.120688 |   0.122690 |   0.130705 |   0.132709 |   0.146718 |
|       8 |   0.114688 |   0.121688 |   0.135708 |   0.136710 |   0.146718 |   0.148719 |   0.162246 |
|       9 |   0.130705 |   0.136710 |   0.150718 |   0.151718 |   0.162246 |   0.164249 |   0.178259 |
|      10 |   0.146718 |   0.150718 |   0.166248 |   0.168250 |   0.178259 |   0.180262 |   0.193275 |
|---------|------------|------------|------------|------------|------------|------------|------------|

min_queue_time = 0
max_queue_time = 0.0070042
avg_queue_time = 0.002668

min_proc_time =  0.0339991
max_proc_time =  0.0480402
avg_proc_time =  0.045978

avg_proc1_time =  0.0145242
avg_proc2_time =  0.00972407
avg_proc3_time =  0.0137257

avg_queue1_time =  0.00460116
avg_queue2_time =  0.00140095
avg_queue3_time =  0.0020019

Without conveyor time:
464698900 ns
464.699 ms
0.464699 s
```

Рис. 4.1: Результат работы программы

Из результатов следует, что больше всего времени в среднем тратится на разбиение строки на слова (1-я лента конвейера), меньше всего – на поиск полиндромов среди них (2-я лента конвейера). Соответственно, время простоя в первой очереди — наибольшее, а во второй — наименьшее. Также, реализация с конвейером быстрее справилась с заявками, чем реализация с последовательной обработкой.

!! либо русские буквы в примере, либо нужны комментарии к обозначениям на рисунке

4.3 Вывод

По итогу иследования выяснилось, что разработанный алгоритм работает верно, то есть находит самый длинный полиндром в строке. Кроме этого был проведён анализ и сделан вывод по логу программы.

Заключение

По итогу проделанной работы была достигнута цель – получен навык организации асинхронной передачи данных между потоками на примере конвейерной обработки информации.

Также были решены все поставленные задачи, а именно:

- выбраны и описаны методы обработки данных, которые сопоставлены методам конвейера;
- реализована конвейерная система, а также сформирован лог событий с указанием времени их происхождения;
- произведено сравнение реализации с конвейером и без.

Список использованных источников

[1] Microsoft developers network – библиотека <chrono>. // URL: https://docs.microsoft.com/ru-ru/cpp/standard-library/chrono?view=msvc-160&viewFallbackFrom=vs-2017.

[2] Как применить настройки оптимизации gcc в qt? // URL: http://blog.kislenko.net/show.php?id=1991.