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»

на тему: «Организация асинхронного взаимодействия потоков вычисления на примере конвейерных вычислений»

Студент ИУ7-54Б (Группа) _____ (Подпись, дата) Булдаков М. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Волкова Л. Л. (И. О. Фамилия)

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сортировка данных является фундаментальной задачей в области информатики и алгоритмов. Независимо от конкретной области применения, эффективные алгоритмы сортировки существенно влияют на производительность программных систем. От правильного выбора алгоритма зависит как время выполнения программы, так и затраты ресурсов компьютера [**knut**].

Алгоритмы сортировки находят применение в следующих сферах:

— базы данных;

— анализ данных и статистика;

— алгоритмы машинного обучения;

— криптография.

Цель данной лабораторной работы — рассмотреть алгоритмы сортировки. Для достижения поставленной цели необходимо выполнить следующие задачи:

— описать алгоритмы гномьей, пирамидальной сортировок и сортировку по алгоритму Шелла;

— разработать программное обеспечение, реализующее алгоритмы сортировок;

— выбрать инструменты для реализации и замера процессорного времени выполнения реализаций алгоритмов;

— проанализировать затраты реализаций алгоритмов по времени.

1 Аналитический раздел

В данном разделе будет представлена информация о многопоточности и исследуемом алгоритме исправления орфографических ошибок в тексте.

1.1 Многопоточность

Многопоточность — это способность центрального процессора одновременно выполнять несколько потоков, используя ресурсы одного процессора. Каждый поток представляет собой последовательность инструкций, которые могут выполняться параллельно с другими потоками, созданными одним и тем же процессом [1].

Процессом называют программу в стадии выполнения [2]. Один процесс может иметь один или несколько потоков. Поток — это часть процесса, которая выполняет задачи, необходимые для выполнения приложения. Процесс завершается, когда все его потоки полностью завершены.

Одной из сложностей, связанных с использованием потоков, является проблема доступа к данным. Основным ограничением является невозможность одновременной записи в одну и ту же ячейку памяти из разных потоков. Это означает, что нужен механизм синхронизации доступа к данным, так называемый «мьютекс» (от англ. mutex — mutual exclusion, взаимное исключение). Мьютекс может быть захвачен одним потоком для работы в режиме монопольного использования или освобожден. Если два потока попытаются захватить мьютекс одновременно, то успех будет у одного потока, а другой будет блокирован, пока мьютекс не освободится.

1.2 Исправления орфографических ошибок в тексте

Для распознавания слов, написанных с ошибками, используется расстояние Левенштейна — минимальное количество ошибок, исправление которых приводит одно слово к другому [3]. Т. о. для введенного слова осуществляется проверка по корпусу, если данное слово не найдено в корпусе, то ищется ближайшее слово к данному по расстоянию Левенштейна.

Кроме того, следует вводить ограничение на количество ошибок, которые позволяется допустить. Как говорит поговорка: «Если в слове хлеб допустить всего четыре ошибки, то получится слово пиво» [3]. Если фиксируется число

ошибок, то для коротких слов оно может оказаться избыточным. Верхнюю границу числа ошибок обычно ограничивают как процентным соотношением, так и фиксированным числом. Например, не более 30% букв входного слова, но не более 3. При этом все равно стараются найти слова с минимальным количеством ошибок [3].

1.3 Использование потоков для исправления орфографических ошибок

Поскольку задача сводится к поиску слова в корпусе, можно распараллелить поиск по этому корпусу. В таком случае каждый поток будет вычислять расстояние Левенштейна между заданным словом и некоторым словом из корпуса и в случае, если расстояние будет удовлетворять требованиям, то данное слово будет записано в массив. При записи подходящих слов в массив, возможна ситуация, когда значение длины массива считывается в одном потоке и в тот же момент изменяется в другом потоке, т. е. возникает конфликт. Для решения проблем синхронизации необходимо использовать мьютекс, чтобы обеспечить монопольный доступ к длине массива.

Вывод

В данном разделе была представлена информация о многопоточности и исследуемом алгоритме.

2 Конструкторский раздел

В этом разделе будет представлено описание используемых типов данных, а также схемы алгоритмов исправления орфографических ошибок.

2.1 Требования к программному обеспечению

Программа должна поддерживать два режима работы: режим массового замера времени и режим исправления введенного слова.

Режим массового замера времени должен обладать следующей функциональностью:

— генерировать корпус слов;

— осуществлять массовый замер, используя сгенерированные данные;

— результаты массового замера должны быть представлены в виде таблицы.

К режиму исправления введенного слова выдвигается следующий ряд требований:

— возможность вводить слова, которые отсутствуют в корпусе;

— слова вводятся на латинице;

— наличие интерфейса для выбора действий;

— на выходе программы, набор из самых близких слов к введенному.

2.2 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры и типы данных:

— слово — массив букв;

— корпус — массив слов, отсортированный в лексикографическом порядке;

— мьютекс — примитив синхронизации.

2.3 Разработка алгоритмов

На рисунке 2.1 представлена схема поиска ближайших слов в корпусе без использования потоков. На рисунке 2.2 представлена схема алгоритма запуска конвейера. На рисунке 2.3 представлена схема алгоритма обслуживающего устройства, которое проверяет содержится ли слово в корпусе. На рисунке 2.4 представлена схема алгоритма обслуживающего устройства, которое находит ближайшие слова. На рисунке 2.5 представлена схема алгоритма обслуживающего устройства, которое записывает слова в файл.

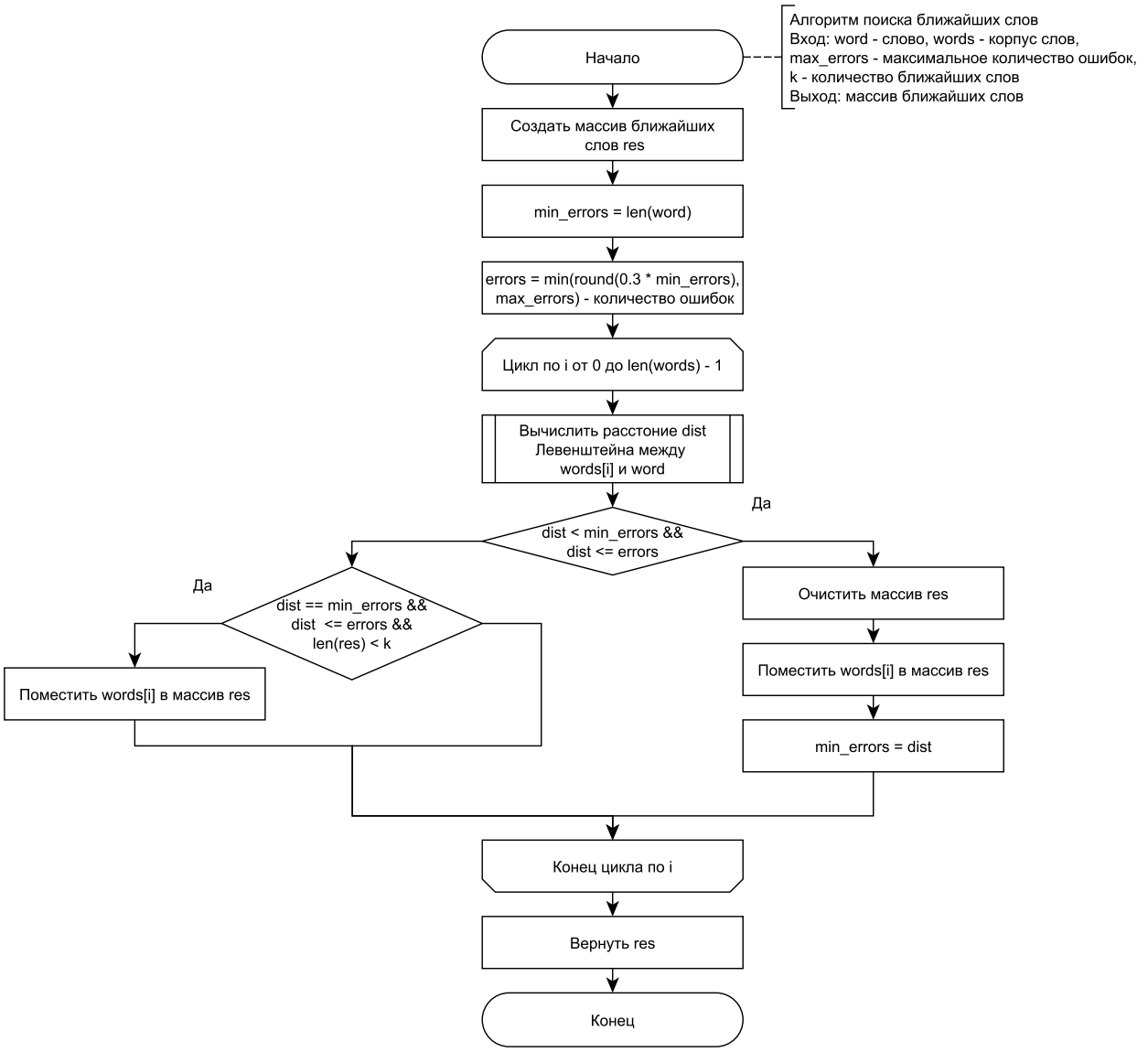

Рисунок 2.1 – Схема алгоритма поиска ближайших слов

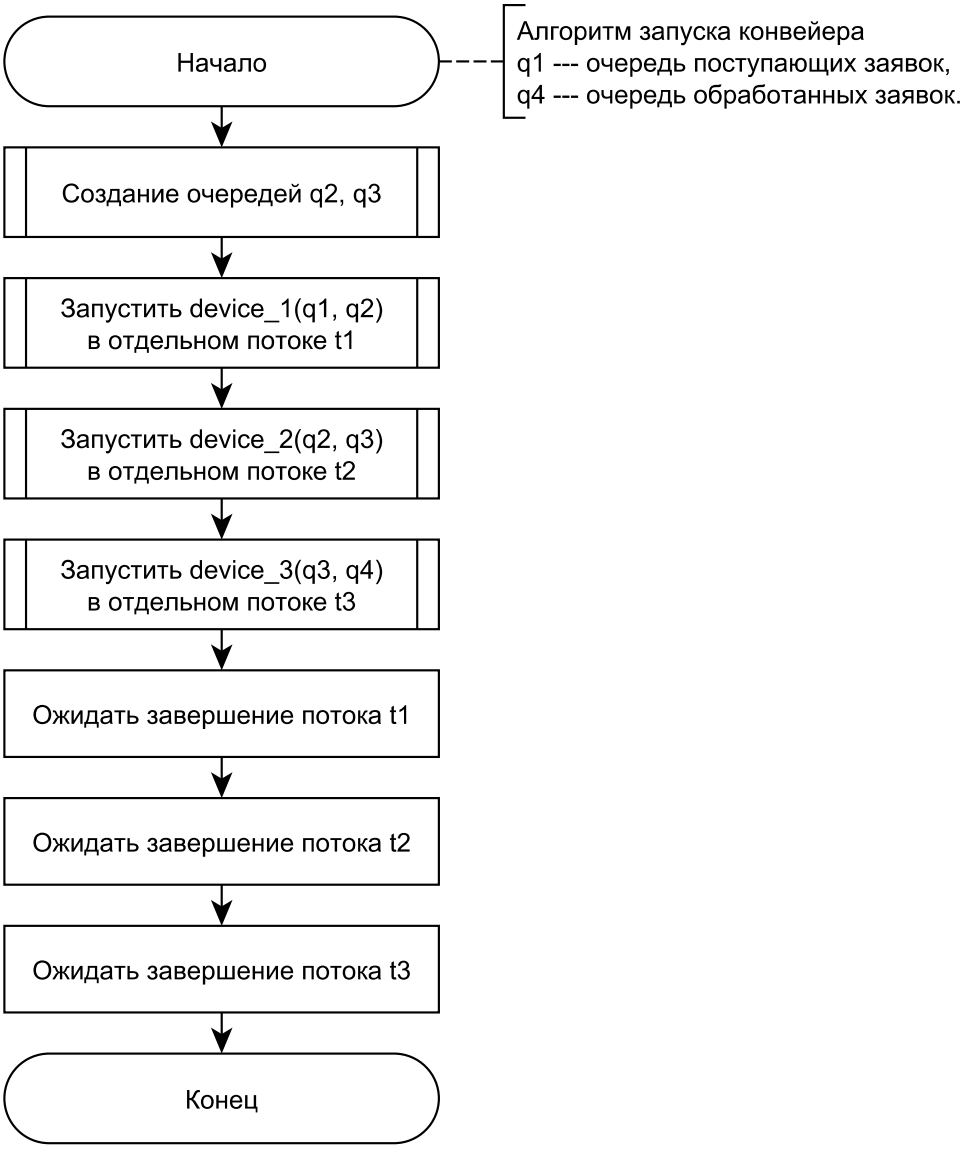

Рисунок 2.2 – Схема алгоритма запуска конвейера

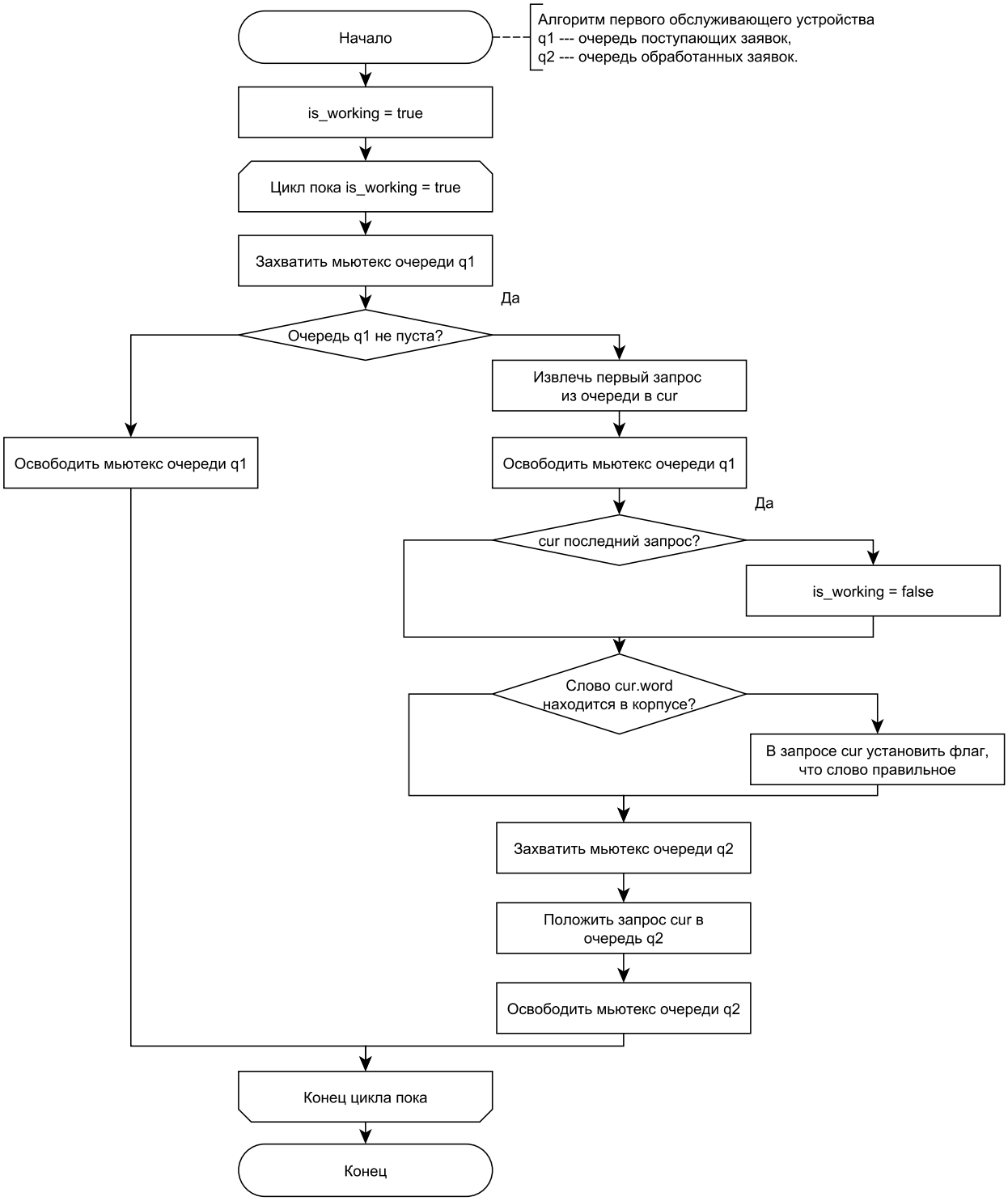

Рисунок 2.3 – Схема алгоритма обслуживающего устройства, которое проверяет содержится ли слово в корпусе

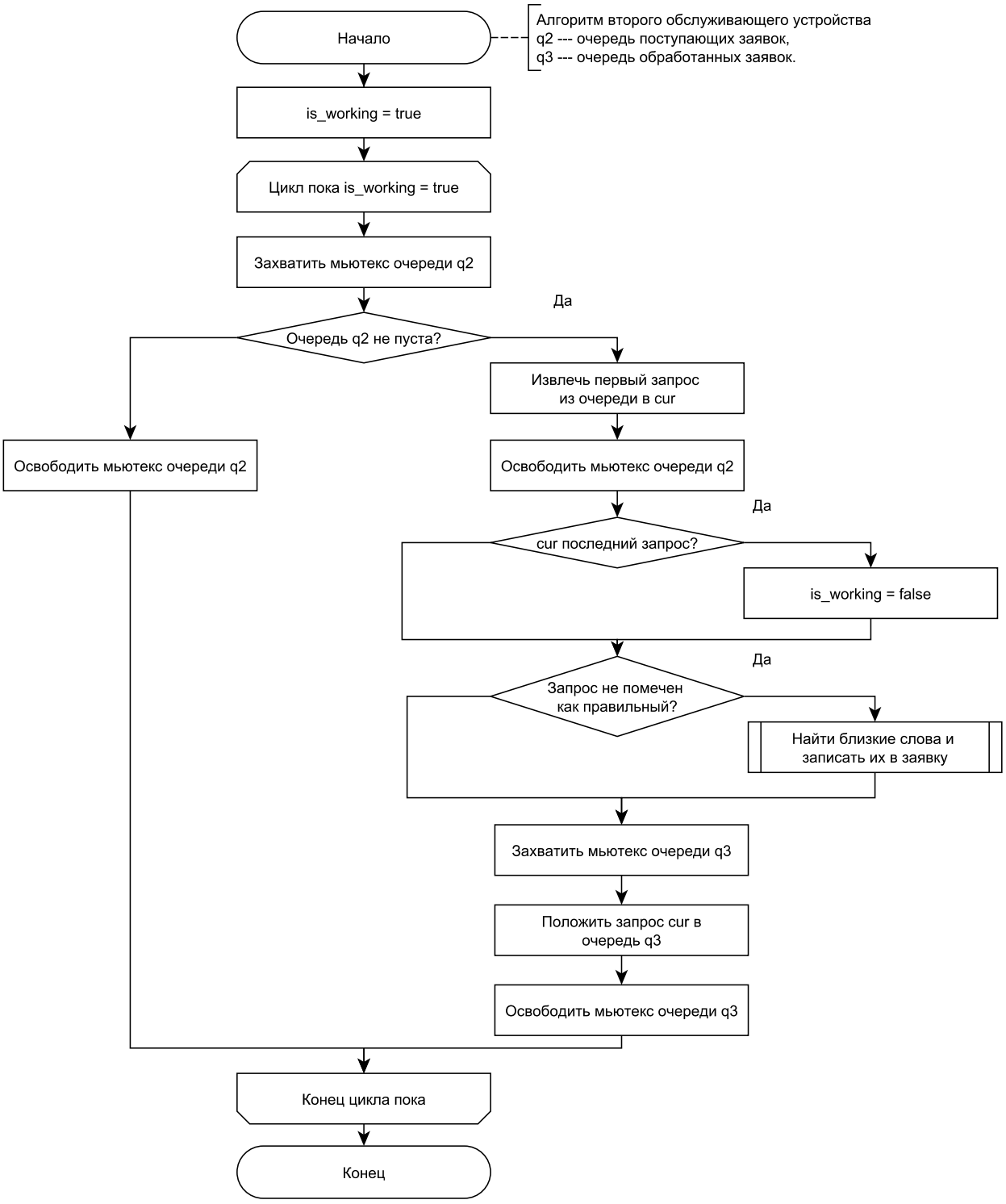

Рисунок 2.4 – Схема алгоритма обслуживающего устройства, которое находит ближайшие слова

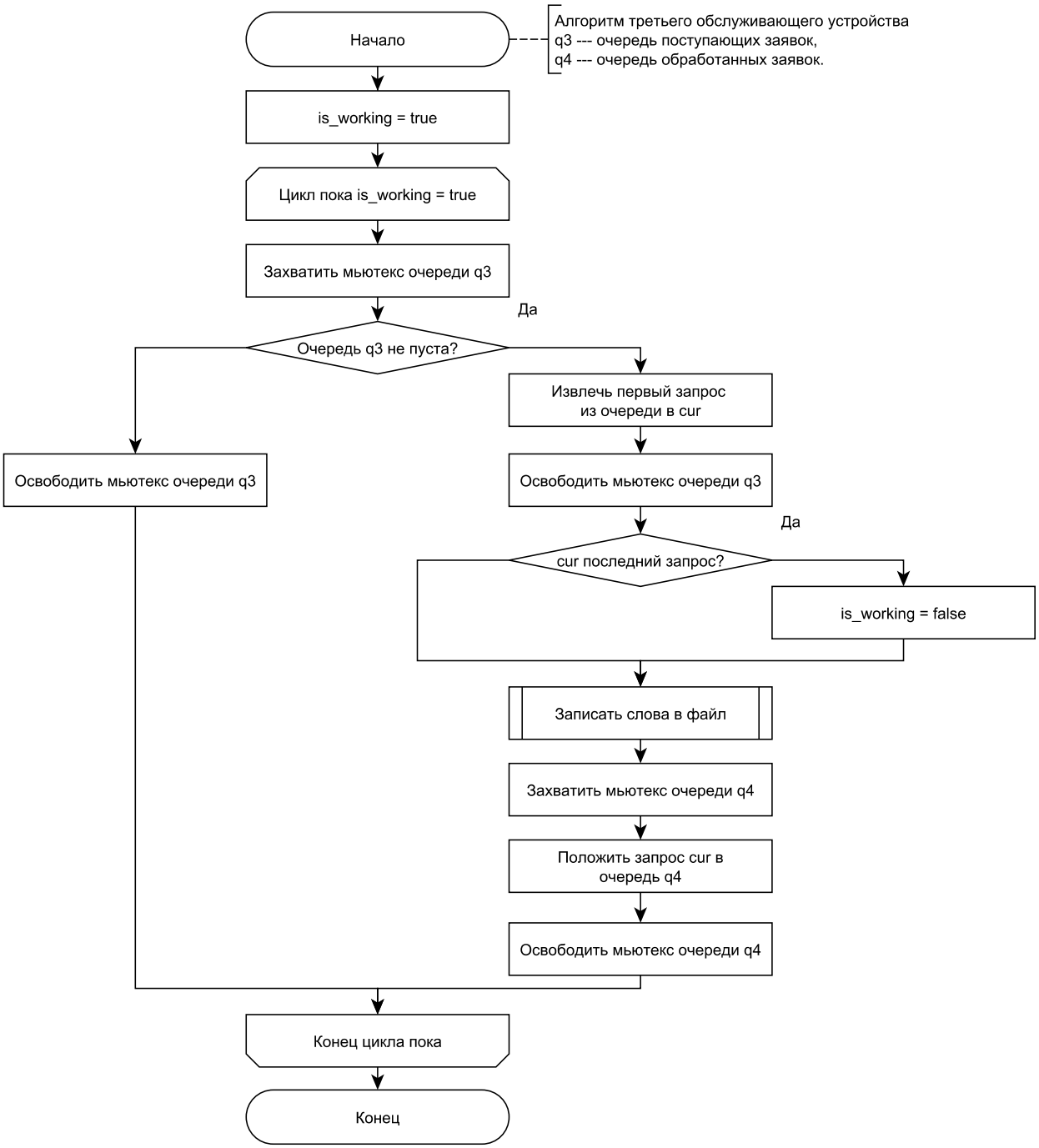

Рисунок 2.5 – Схема алгоритма обслуживающего устройства, которое записывает слова в файл

Вывод

На основе теоретических данных, полученных из аналитического раздела были построены схемы требуемых алгоритмов.

3 Технологический раздел

В данном разделе будут приведены требования к программному обеспечению, средства реализации, листинг кода и функциональные тесты.

3.1 Средства реализации

Для реализации данной работы был выбран язык *C++* [4]. Данный выбор обусловлен следующим:

— язык поддерживает все структуры данных, которые выбраны в результате проектирования;

— язык позволяет реализовать все алгоритмы, выбранные в результате проектирования;

— язык позволяет работать с нативными потоками [5].

Время выполнения реализаций было замерено с помощью функции *clock* [6]. Для хранения слов использовалась структура данных *string* [7], в качестве массивов использовалась структура данных *vector* [8]. В качестве примитива синхронизации использовался *mutex* [9].

Для создания потоков и работы с ними был использован класс *thread* из стандартной библиотеки выбранного языка [5]. В листинге 3.1, приведен пример работы с описанным классом, каждый объект класса представляет собой поток операционной системы, что позволяет нескольким функциям выполняться параллельно [5].

Листинг 3.1 – Пример работы с классом thread

```
#include <iostream>
#include <thread>

void foo(int a)
{
    std::cout << a << '\n';
}

int main()
{
    std::thread thread(foo, 10);
    thread.join();

    return 0;
}
```

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

— *main.cpp* — файл, содержащий функцию *main*;

— *correcter.cpp* — файл, содержащий код реализации алгоритма исправления ошибок;

— *utils.cpp* — файл, в котором содержатся вспомогательные функции;

— *conveyor.cpp* — файл, в котором содержатся реализации элементов конвейера;

— *levenstein.cpp* — файл, в котором содержится реализация алгоритма поиска расстояния Левенштейна.

3.3 Реализация алгоритмов

В листинге 3.2 приведена реализация алгоритма исправления ошибок без дополнительных потоков. В листинге 3.3 приведена реализация алгоритма запуска конвейера. В листингах 3.4 – 3.6 приведены реализации обслуживающих устройств.

Листинг 3.2 – Функция исправления ошибок

```
std::vector<std::wstring> get_closest_words(
    const std::vector<std::wstring> &words,
    const std::wstring &word, size_t k, size_t max_errors) {

    std::vector<std::wstring> temp;
    size_t min = word.size();

    size_t errors = std::min(static_cast<size_t>(std::ceil(0.3 *
       word.size())), max_errors);

    for (const auto &cur_word: words) {
        int dist = lev_mtr(cur_word, word);
        if (dist < min && dist <= errors) {
            temp.clear();
            temp.push_back(cur_word);
            min = dist;
        } else if (dist == min && dist <= errors && temp.size()
           < k)
            temp.push_back(cur_word);
    }

    return temp;
}
```

Листинг 3.3 – Функция запуска конвейера

```
void run_pipeline(AtomicQueue<Request> &start,
                  AtomicQueue<Request> &end,
                  const std::string &fname_in,
                  const std::string &fname_out) {

    auto words = read_words_from_file(fname_in);

    AtomicQueue<Request> secondQ;
    AtomicQueue<Request> thirdQ;

    std::thread t1(device1, std::ref(start), std::ref(secondQ),
       std::ref(words));
    std::thread t2(device2, std::ref(secondQ), std::ref(thirdQ),
       std::ref(words));
    std::thread t3(device3, std::ref(thirdQ), std::ref(end),
       fname_out);
    t1.join();
    t2.join();
    t3.join();
}
```

Листинг 3.4 – Функция обслуживающего устройства, которое проверяет содержится ли слово в корпусе

```
void device1(AtomicQueue<Request> &from, AtomicQueue<Request>
   &to, const std::vector<std::wstring> &words) {
    bool is_working = true;

    while (is_working) {
        if (from.size() > 0) {
            timespec start, end;
            Request cur_request = from.front();

            if (cur_request.is_last)
                is_working = false;

            from.pop();

            start = get_time();
            if (is_word_in_vec(words, cur_request.word))
                cur_request.is_correct = true;
            end = get_time();

            cur_request.time_start_1 = start;
            cur_request.time_end_1 = end;
            to.push(cur_request);
        }

    }
}
```

Листинг 3.5 – Функция обслуживающего устройства, которое находит ближайшие слова

```
1
2  void device2(AtomicQueue<Request> &from, AtomicQueue<Request>
       &to, const std::vector<std::wstring> &words) {
3      bool is_working = true;
4
5      while (is_working) {
6          if (from.size() > 0) {
7              timespec start, end;
8              Request cur_request = from.front();
9
10             if (cur_request.is_last)
11                 is_working = false;
12
13             from.pop();
14             start = get_time();
15             if (!cur_request.is_correct)
16                 cur_request.res = get_closest_words(words,
17                                                     cur_request.word,
18                                                     cur_request.k,
19                                                     cur_request.max_e
20             end = get_time();
21             cur_request.time_start_2 = start;
22             cur_request.time_end_2 = end;
23             to.push(cur_request);
24         }
25     }
26 }
```

Листинг 3.6 – Функция обслуживающего устройства, которое записывает слова в файл

```
void device3(AtomicQueue<Request> &from, AtomicQueue<Request>
   &to, const std::string &fname) {
    bool is_working = true;

    while (is_working) {
        if (from.size() > 0) {
            timespec start, end;
            Request cur_request = from.front();

            if (cur_request.is_last)
                is_working = false;

            from.pop();
            start = get_time();
            print_to_file(cur_request, fname);
            end = get_time();
            cur_request.time_start_3 = start;
            cur_request.time_end_3 = end;
            to.push(cur_request);
        }
    }
}
```

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для разработанных алгоритмов исправления ошибок. Для данных тестов максимальное количество ошибок равно двум и из массива выбираются 3 слова. Все тесты пройдены успешно. В таблице 3.1 пустое слово обозначается с помощью λ.

Таблица 3.1 – Функциональные тесты

Корпус	**Слово**	**Ожидаемый результат**
[мама, мыла, раму]	мама	[мама]
[мама, мыла, раму]	мамы	[мама]
[мама, мыла, раму]	мыма	[мама, мыла]
[мама, мыла, раму]	ахтунг	[ ]
[ ]	ахтунг	[ ]
[ ]	λ	[ ]
[Мама, Мыла, Раму]	λ	[ ]

Вывод

Были разработаны и протестированы спроектированные алгоритмы исправления ошибок.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Stoltzfus J.* Multithreading [Электронный ресурс]. — Режим доступа: https://www.techopedia.com/definition/24297/multithreading-computer-architecture (дата обращения: 07.12.2023).
2. *Ричард Стивенс У.*, *Стивен Раго А.* UNIX. Профессиональное программирование. 3-е издание. — СПб.: Питер, 2018. — С. 994.
3. Автоматическая обработка текстов на естественном языке и компьютерная лингвистика / Е. Большакова [и др.] //. — М.: МИЭМ, 2011. — С. 122—124.
4. C++ reference [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/ (дата обращения: 20.12.2023).
5. Concurrency support library [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/thread (дата обращения: 20.12.2023).
6. std::clock [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/chrono/c/clock (дата обращения: 20.12.2023).
7. Strings library [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/string (дата обращения: 20.12.2023).
8. std::vector [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/string (дата обращения: 20.12.2023).
9. std::mutex [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/thread/mutex (дата обращения: 20.12.2023).