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

С развитием вычислительных систем появилась потребность в параллельной обработке данных для повышения эффективности систем, ускорения вычислений и более рационального использования имеющихся ресурсов. Благодаря совершенствованию процессоров стало возможно использовать их для выполнения множества одновременных задач, что привело к появлению понятия «многопоточность» [1].

Цель данной лабораторной работы — описать принципы конвейерных вычислений на основе нативных потоков для исправления орфографических ошибок в тексте. Для достижения поставленной цели необходимо выполнить следующие задачи:

— описать алгоритм исправления орфографических ошибок в тексте;

— спроектировать программное обеспечение, реализующее алгоритм и его конвейерную версию;

— 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 [2].

Процессом называют программу в стадии выполнения [3]. 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 [4]. 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» [4]. 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 [4].

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 запуска конвейера.

Режим массового замера времени должен обладать следующей функциональностью:

— генерировать очередь заявок;

— осуществлять массовый замер, используя сгенерированные данные;

— результаты массового замера должны быть представлены в виде таблицы.

К режиму запуска конвейера выдвигается следующий ряд требований:

— ввод числа заявок;

— генерировать очередь заявок;

— представить результат в виде файла.

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.

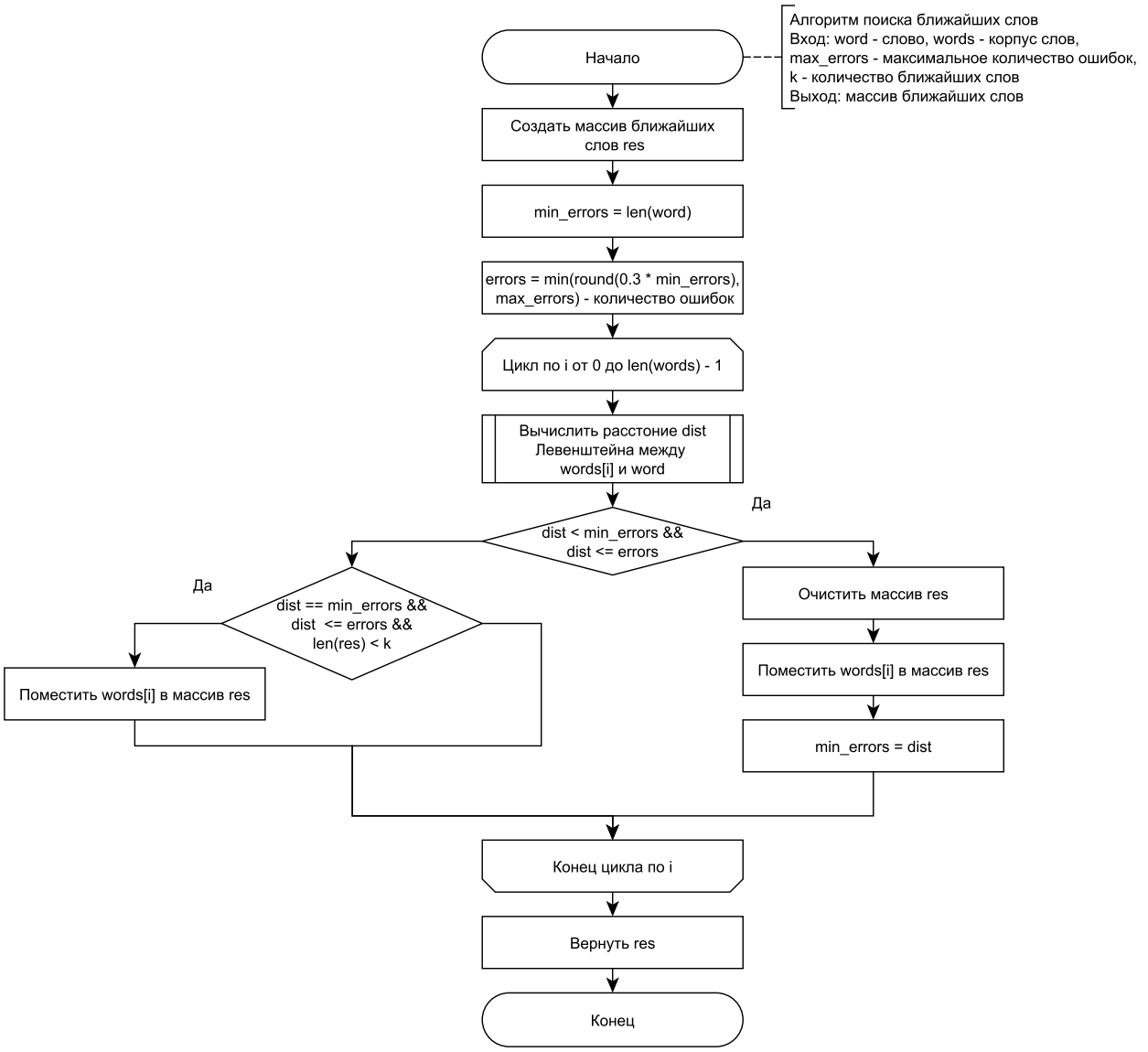

Рисунок 2.1 – Схема алгоритма поиска ближайших слов

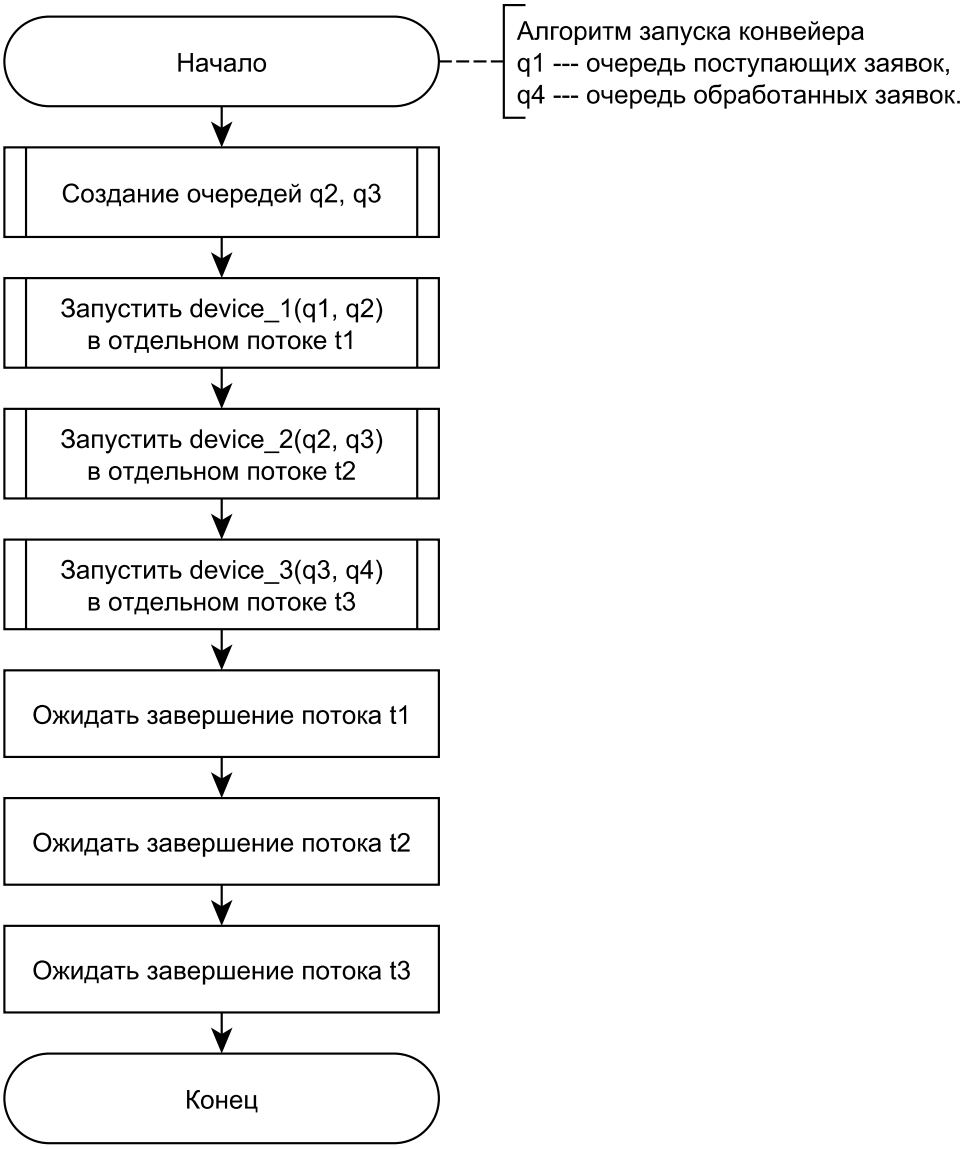

Рисунок 2.2 – Схема алгоритма запуска конвейера

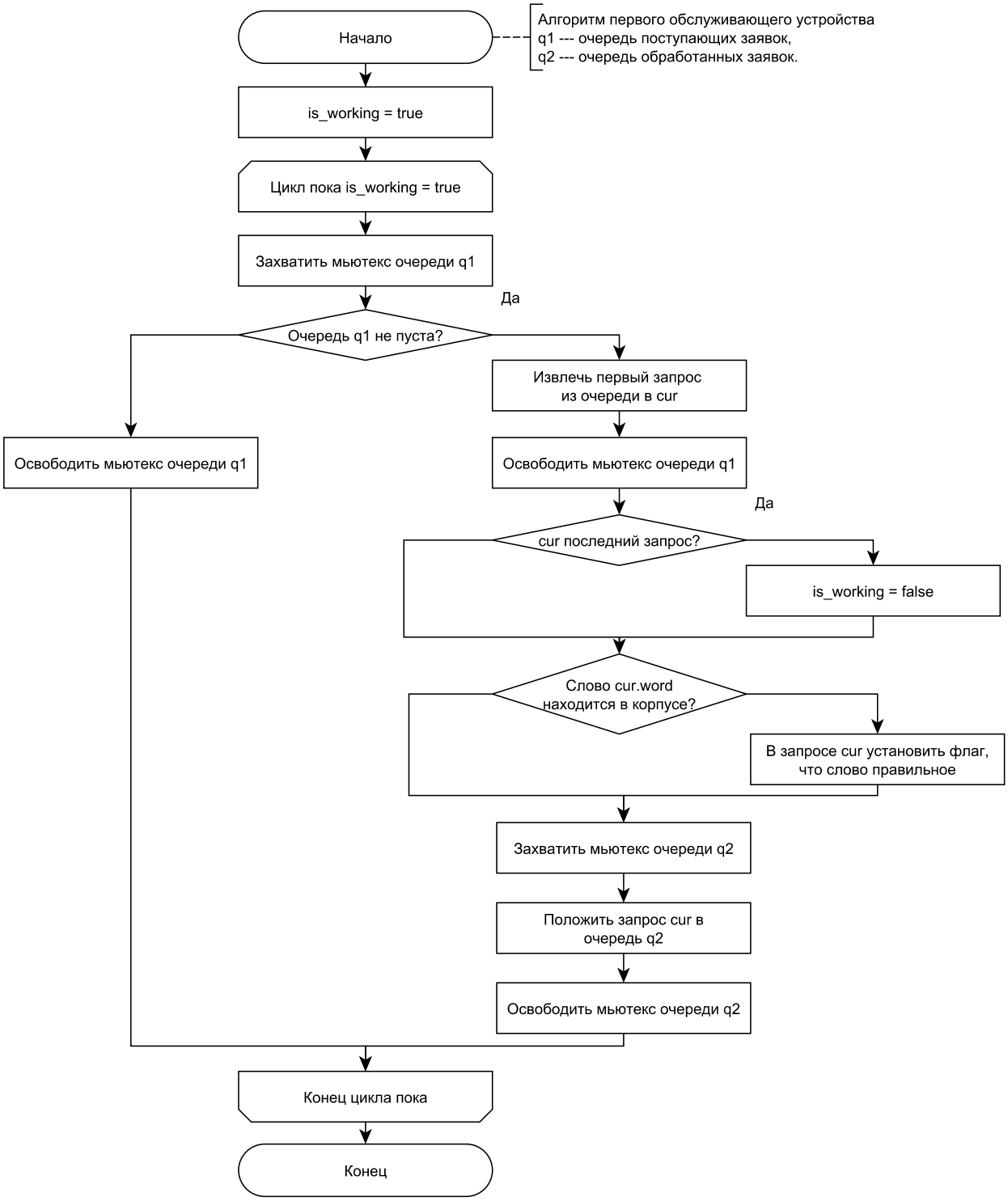

Рисунок 2.3 – Схема алгоритма обслуживающего устройства, которое проверяет содержится ли слово в корпусе

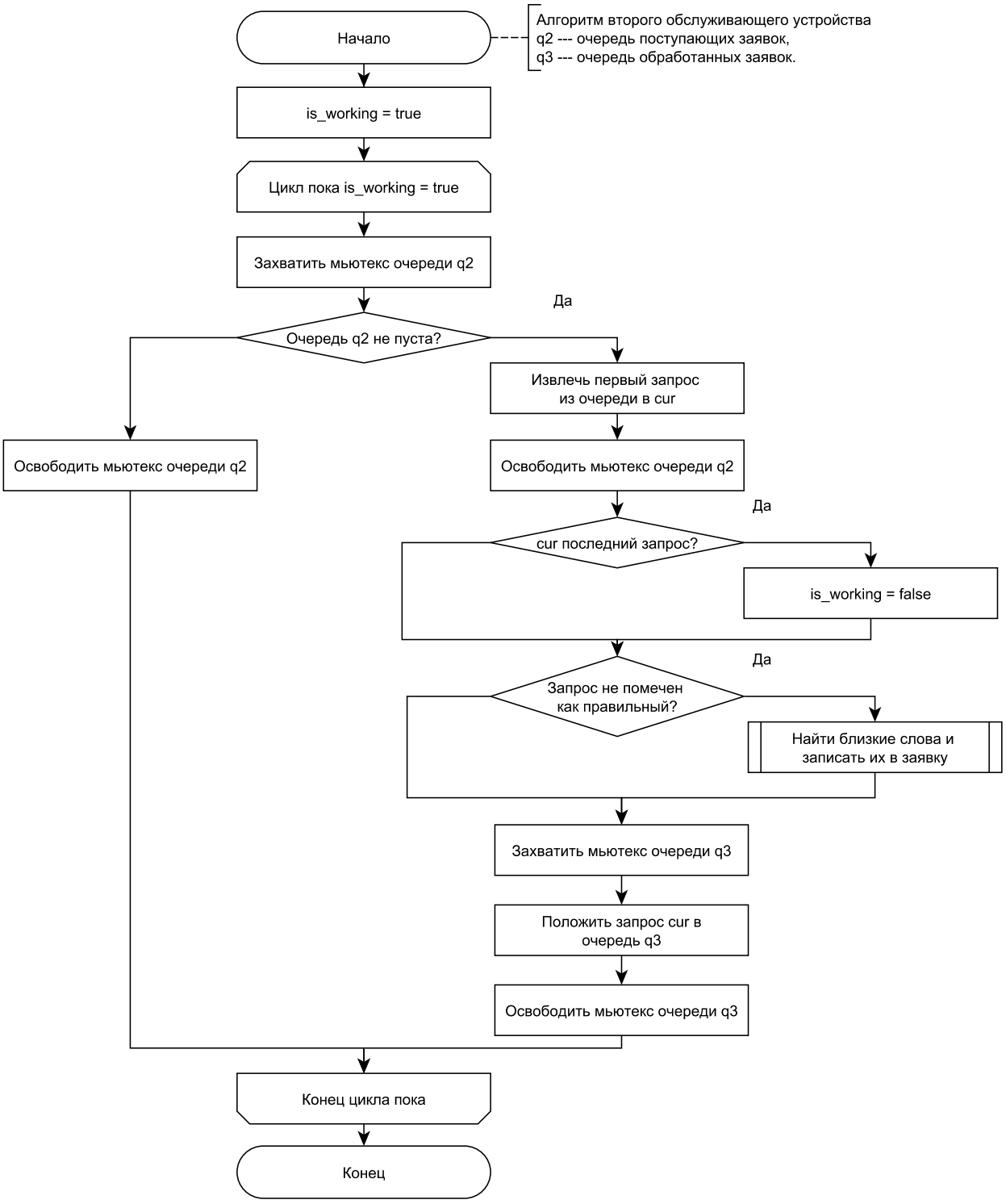

Рисунок 2.4 – Схема алгоритма обслуживающего устройства, которое находит ближайшие слова

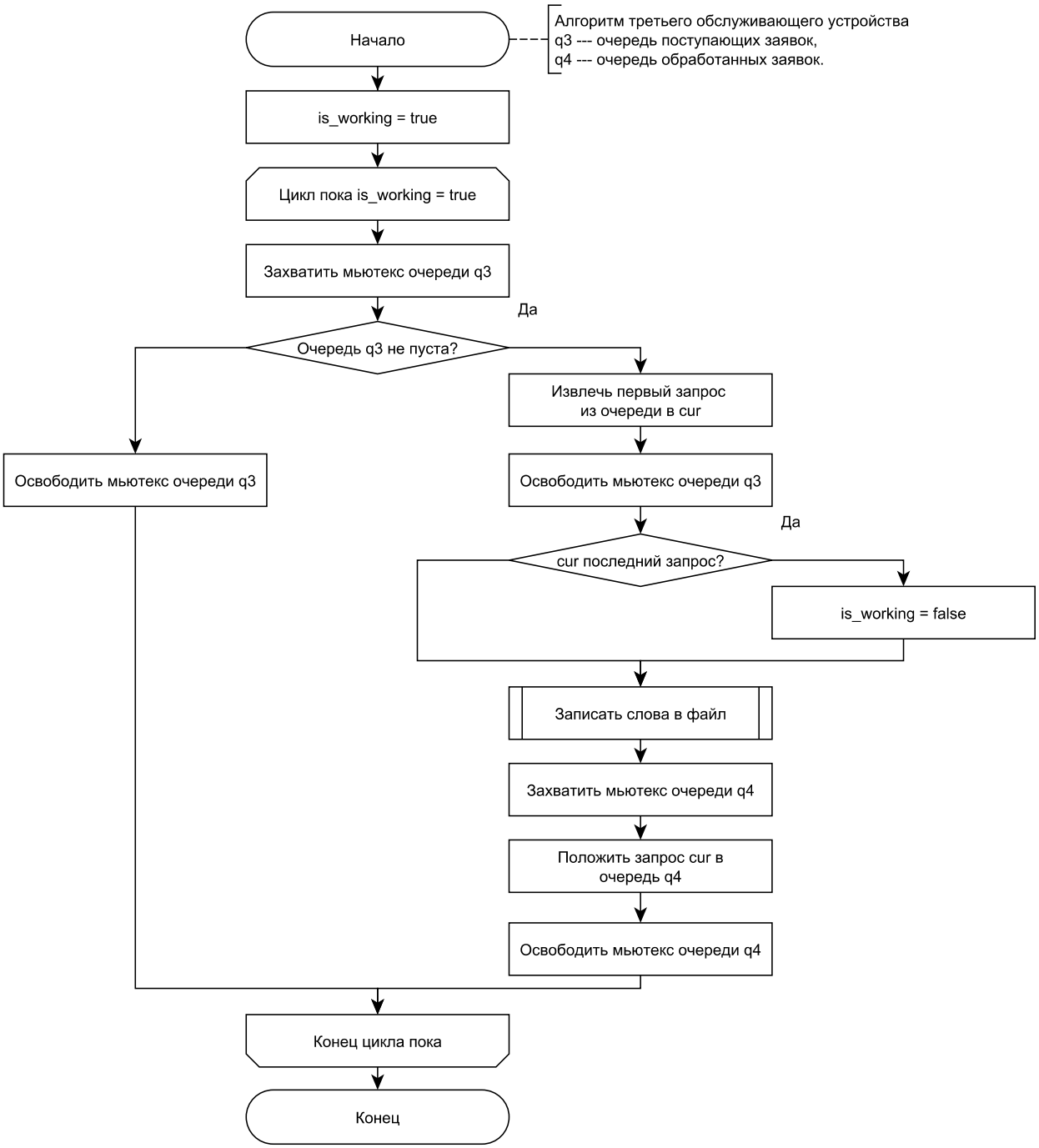

Рисунок 2.5 – Схема алгоритма обслуживающего устройства, которое записывает слова в файл

Вывод

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

3 Технологический раздел

В данном разделе будут приведены средства реализации и листинг кода.

3.1 Средства реализации

Для реализации данной работы был выбран язык *C++* [5]. 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 [6].

Время выполнения реализаций было замерено с помощью функции *clock* [7]. Для хранения слов использовалась структура данных *wstring* [8], в качестве массивов использовалась структура данных *vector* [9]. В качестве примитива синхронизации использовался *mutex* [10].

Для создания потоков и работы с ними был использован класс *thread* из стандартной библиотеки выбранного языка [6]. В листинге 3.1, приведен пример работы с описанным классом, каждый объект класса представляет собой поток операционной системы, что позволяет нескольким функциям выполняться параллельно [6].

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);
        }
    }
}
```

Вывод

Были разработаны спроектированные алгоритмы исправления ошибок.

4 Исследовательский раздел

В данном разделе будут приведены: пример работы программы, постановка исследования и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Демонстрация работы программы

На рисунке 4.1 представлена демонстрация работы разработанного программного обеспечения, а именно показаны результаты генерации и конвейерной обработки 5 слов.

```
Меню:
1 - Запустить конвейер
2 - Замерить время
3 - Вывести статистику
0 - Выйти

Выберите пункт меню:1
 1
Введит количество заявок:5
 5
лохлан  лохман
туловищб        туловища
ппироде природе
мов     мог
Меню:
1 - Запустить конвейер
2 - Замерить время
3 - Вывести статистику
0 - Выйти
```

Рисунок 4.1 – Демонстрация работы программы

Технические характеристики устройства, на котором выполнялись замеры по времени, следующие:

— процессор: AMD Ryzen 5 4600H 3 ГГц, 6 физических ядер, 12 логических процессоров [11];

— оперативная память: 16 ГБайт;

— операционная система: Windows 10 Pro 64-разрядная система версии 22H2 [12].

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Время выполнения реализаций алгоритмов

Результаты замеров времени выполнения реализаций приведены в таблице 4.1. Замеры времени проводились случайно сгенерированных словах. Замеры времени проводились на корпусах одного размера.

Таблица 4.1 – Время работы реализаций алгоритмов (в с)

Количество слов	Поточная (с)	Линейная (с)
10	$1.411 \cdot 10^{-2}$	$3.488 \cdot 10^{-2}$
20	$2.715 \cdot 10^{-2}$	$5.926 \cdot 10^{-2}$
30	$4.851 \cdot 10^{-2}$	$9.296 \cdot 10^{-2}$
40	$6.160 \cdot 10^{-2}$	$1.224 \cdot 10^{-1}$
50	$7.296 \cdot 10^{-2}$	$1.292 \cdot 10^{-1}$
60	$9.610 \cdot 10^{-2}$	$1.836 \cdot 10^{-1}$
70	$1.428 \cdot 10^{-1}$	$2.461 \cdot 10^{-1}$

На рисунке 4.2 приведен график зависимости времени выполнения реализаций от количества заявок.

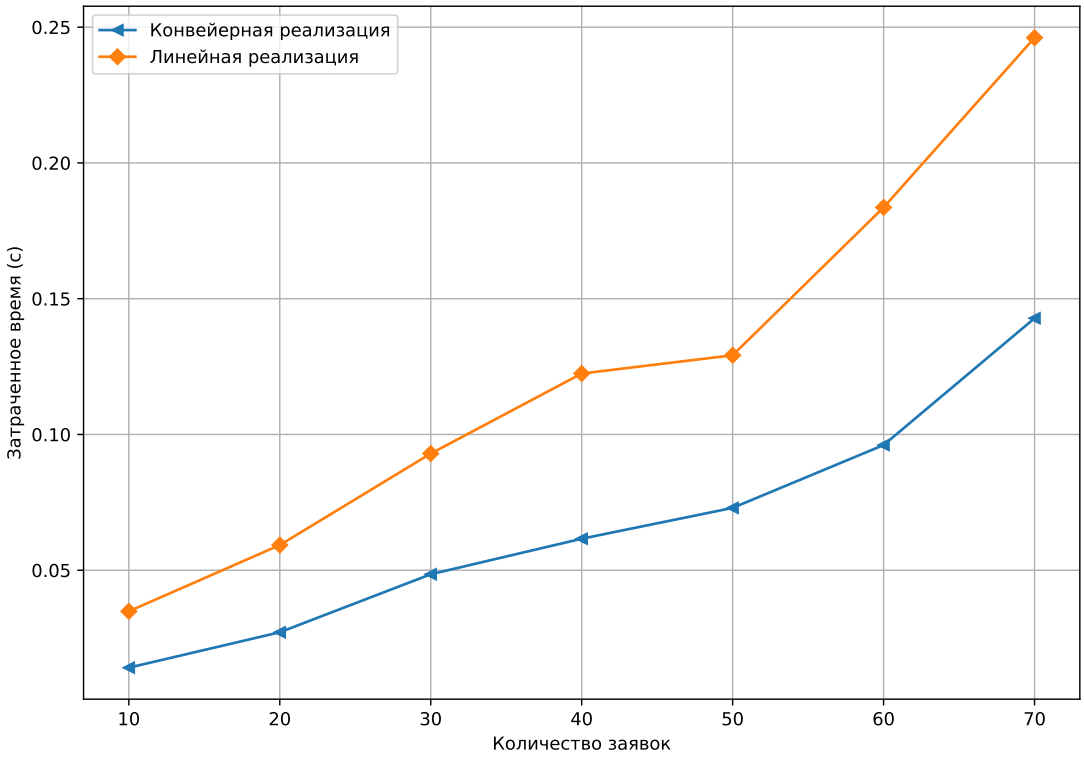

Рисунок 4.2 – График зависимости времени выполнения реализации от количества заявок

Вывод

В результате анализа таблицы 4.1, было получено, что время выполнения поточной реализации в 2 раза быстрее линейной реализации, при количестве заявок 70. Такой результат объясняется тем, что в поточной реализации потоки могут выполнять различные этапы работы параллельно, что позволяет сократить время обработки последовательности заявок.

ЗАКЛЮЧЕНИЕ

Цель данной лабораторной работы была достигнута, а именно описаны принципы конвейерных вычислений на основе нативных потоков для исправления орфографических ошибок в тексте

Для достижения поставленной цели были выполнены следующие задачи:

— описан алгоритм исправления орфографических ошибок в тексте;

— спроектировано программное обеспечение, реализующее алгоритм и его конвейерную версию;

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

— проанализированы затраты реализаций алгоритмов по времени.

В результате исследования реализаций было получено, было получено, что время выполнения поточной реализации в 2 раза быстрее линейной реализации, при количестве заявок 70. Такой результат объясняется тем, что в поточной реализации потоки могут выполнять различные этапы работы параллельно, что позволяет сократить время обработки последовательности заявок.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Гладышев Е.*, *Мурыгин А.* Многопоточность в приложениях // Актуальные проблемы авиации и космонавтики. — 2012. — № 8.
2. *Stoltzfus J.* Multithreading [Электронный ресурс]. — Режим доступа: `https://www.techopedia.com/definition/24297/multithreading-computer-architecture` (дата обращения: 07.12.2023).
3. *Ричард Стивенс У.*, *Стивен Раго А.* UNIX. Профессиональное программирование. 3-е издание. — СПб.: Питер, 2018. — С. 994.
4. Автоматическая обработка текстов на естественном языке и компьютерная лингвистика / Е. Большакова [и др.] //. — М.: МИЭМ, 2011. — С. 122—124.
5. C++ reference [Электронный ресурс]. — Режим доступа: `https://en.cppreference.com/w/` (дата обращения: 20.12.2023).
6. Concurrency support library [Электронный ресурс]. — Режим доступа: `https://en.cppreference.com/w/cpp/thread` (дата обращения: 20.12.2023).
7. std::clock [Электронный ресурс]. — Режим доступа: `https://en.cppreference.com/w/cpp/chrono/c/clock` (дата обращения: 20.12.2023).
8. Strings library [Электронный ресурс]. — Режим доступа: `https://en.cppreference.com/w/cpp/string` (дата обращения: 20.12.2023).
9. std::vector [Электронный ресурс]. — Режим доступа: `https://en.cppreference.com/w/cpp/string` (дата обращения: 20.12.2023).
10. std::mutex [Электронный ресурс]. — Режим доступа: `https://en.cppreference.com/w/cpp/thread/mutex` (дата обращения: 20.12.2023).
11. Amd [Электронный ресурс]. — Режим доступа: `https://www.amd.com/en.html` (дата обращения: 20.12.2023).
12. Windows 10 Pro 22h2 64-bit [Электронный ресурс]. — Режим доступа: `https://www.microsoft.com/ru-ru/software-download/windows10` (дата обращения: 20.12.2023).