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 дисциплине «Анализ алгоритмов»

Тема Конвейерная обработка данных

Студент Шматко К. М.

Группа ИУ7-56Б

Оценка (баллы) _____

Преподаватель: Волкова Л. Л.

Москва — 2023 г.

Содержание

Введение

При обработке данных могут возникать ситуации, когда один набор данных необходимо обработать последовательно несколькими алгоритмами. В таком случае удобно использовать конвейерную обработку данных, что позволяет на каждой следующей «линии» конвейера использовать данные, полученные с предыдущего этапа. Отдельно стоит упомянуть асинхронные конвейерные вычисления. Отличие от линейных состоит в том, что при таком подходе линии работают с меньшим временем простоя, так как могут обрабатывать задачи независимо от других линий.

Целью данной лабораторной работы является исследование конвейерных вычислений.

Для достижения поставленной цели необходимо выполнить следующие задачи.

— Описать организацию конвейерной обработки данных.

— Описать алгоритмы обработки данных, которые будут использоваться в текущей лабораторной работе.

— Реализовать программу, выполняющую конвейерную обработку с количеством лент не менее трех.

— Проанализировать реализацию алгоритма по затраченному времени.

1 Аналитическая часть

В этом разделе будет представлена информация об основах конвейерной обработки данных и исследуемом алгоритме получения гипотез для неоднозначных словоформ.

1.1 Конвейерная обработка данных

Конвейер (или конвейеризация) представляет собой метод организации вычислений, который позволяет увеличить количество выполняемых инструкций в единицу времени за счет использования принципов параллельной обработки [1]. Этот подход в общем случае основан на разделении функции, предназначенной для выполнения, на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры. Конвейеризация позволяет разбить обработку машинных команд на несколько этапов, организовав передачу данных от одного этапа к следующему. Такой метод позволяет совмещать выполнение различных команд и повышает производительность, поскольку на различных ступенях конвейера одновременно выполняется несколько команд.

Хотя конвейеризация увеличивает пропускную способность процессора (количество команд, завершающихся в единицу времени), важно отметить, что она не сокращает время выполнения отдельной команды. Фактически, она даже немного увеличивает время выполнения каждой команды из-за дополнительных расходов, связанных с хранением промежуточных результатов. Тем не менее, увеличение пропускной способности приводит к более быстрому выполнению программы по сравнению с простыми не конвейерными схемами.

1.2 Библиотека *pymorphy2*

Библиотека *pymorphy2* — морфологический анализатор, написанный на языке Python [2]. Он умеет следующее.

1) Приводить слово к нормальной форме (например, «люди – человек», или «гулял – гулять»).

2) Ставить слово в нужную форму. Например, ставить слово во множественное число, менять падеж слова и т.д.

3) Возвращать грамматическую информацию о слове (число, род, падеж, часть речи и т.д.).

В *pymorphy2* для морфологического анализа слов есть класс *MorphAnalyzer*. Метод *MorphAnalyzer.parse*() возвращает один или несколько объектов типа *Parse* с информацией о том, как слово может быть разобрано. Для получения формологических свойств словоформы используется тег – набор граммем, характеризующих данное слово.

Словоформы для которых возвращается несколько вариантов гипотез являются неоднозначными.

1.3 Описание алгоритмов конвейера

В качестве операций, выполняющихся на конвейере, взяты следующие:

1) считывание словоформ из файла;

2) получение гипотез для словоформ;

3) запись гипотез в файл.

Обработчики будут передавать друг другу заявки.

Заявка будет содержать:

— массив словоформ;

— массив гипотез;

— номер заявки;

— имя файла;

— временные отметки начала и конца выполнения стадии обработки заявки.

Вывод

В данном разделе была представлена информация об основах конвейерной обработки данных и исследуемом алгоритме.

2 Конструкторская часть

В данном разделе будут приведены схемы алгоритма получения гипотез для неоднозначных словоформ в конвейерной реализации.

2.1 Разработка алгоритмов

На рисунке 2.1 представлена схема главного потока конвейера. На рисунках 2.2–2.4 представлены схемы алгоритмов каждого из обработчиков (потоков). На рисунке 2.5 представлена схема алгоритма получения гипотез для неоднозначных словоформ.

Так как первый и второй обработчики и второй и третий обработчики имеют общие ресурсы в виде очереди, то для работы с данными критическими секциями необходимо использовать такой примитив синхронизации как мьютекс [3].

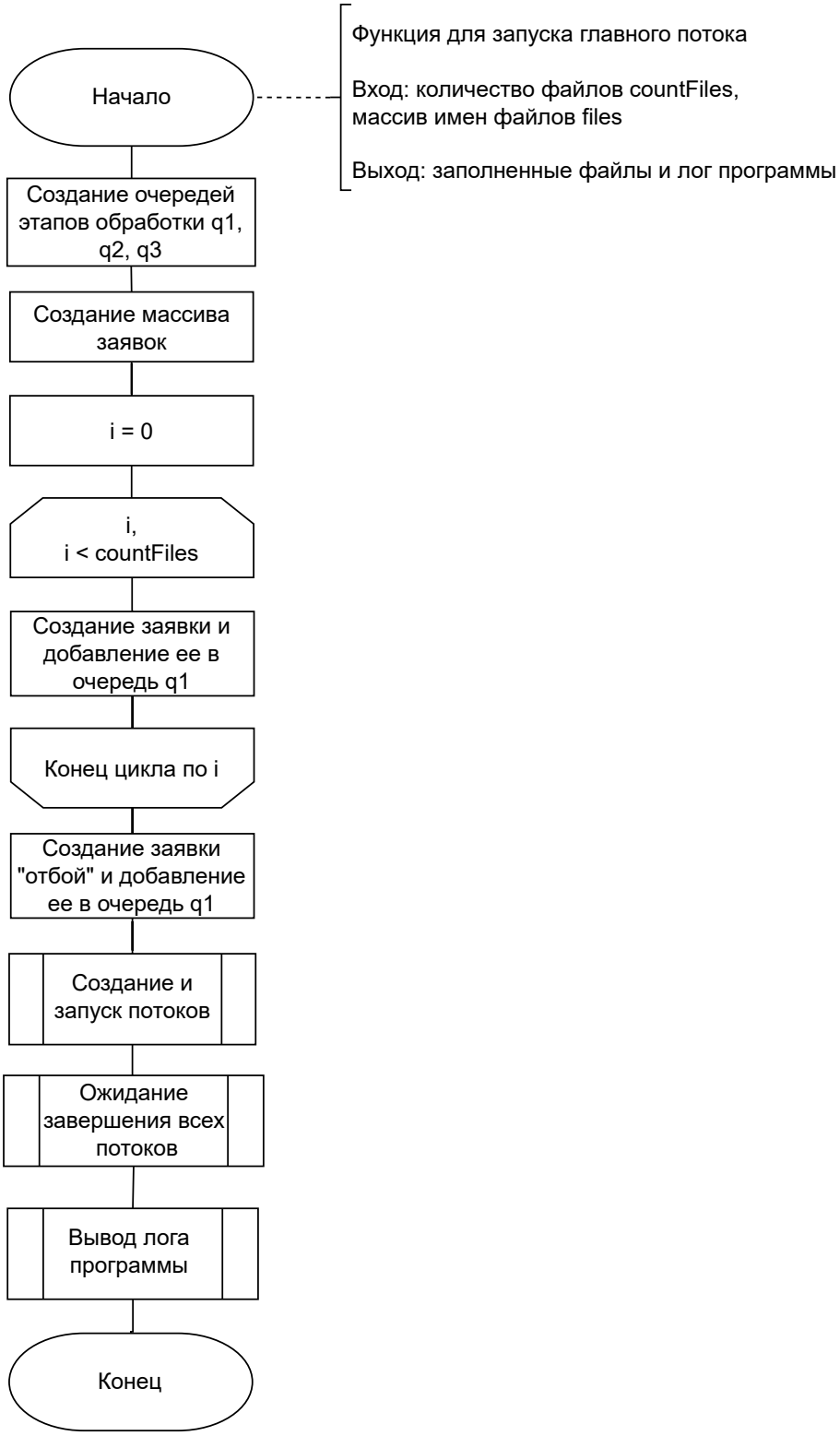

Рисунок 2.1 – Схема алгоритма главного потока конвейера

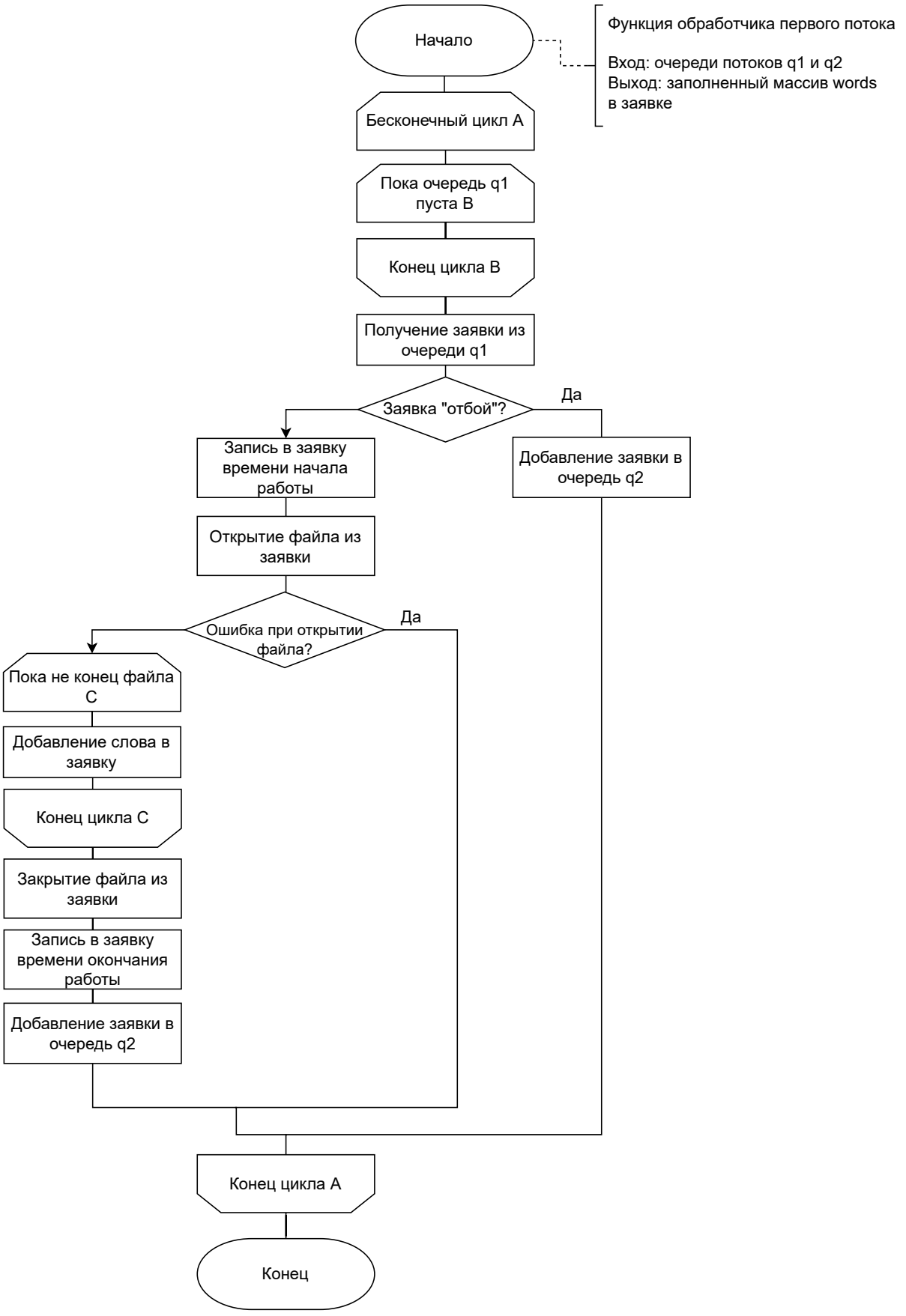

Рисунок 2.2 – Схема алгоритма потока 1

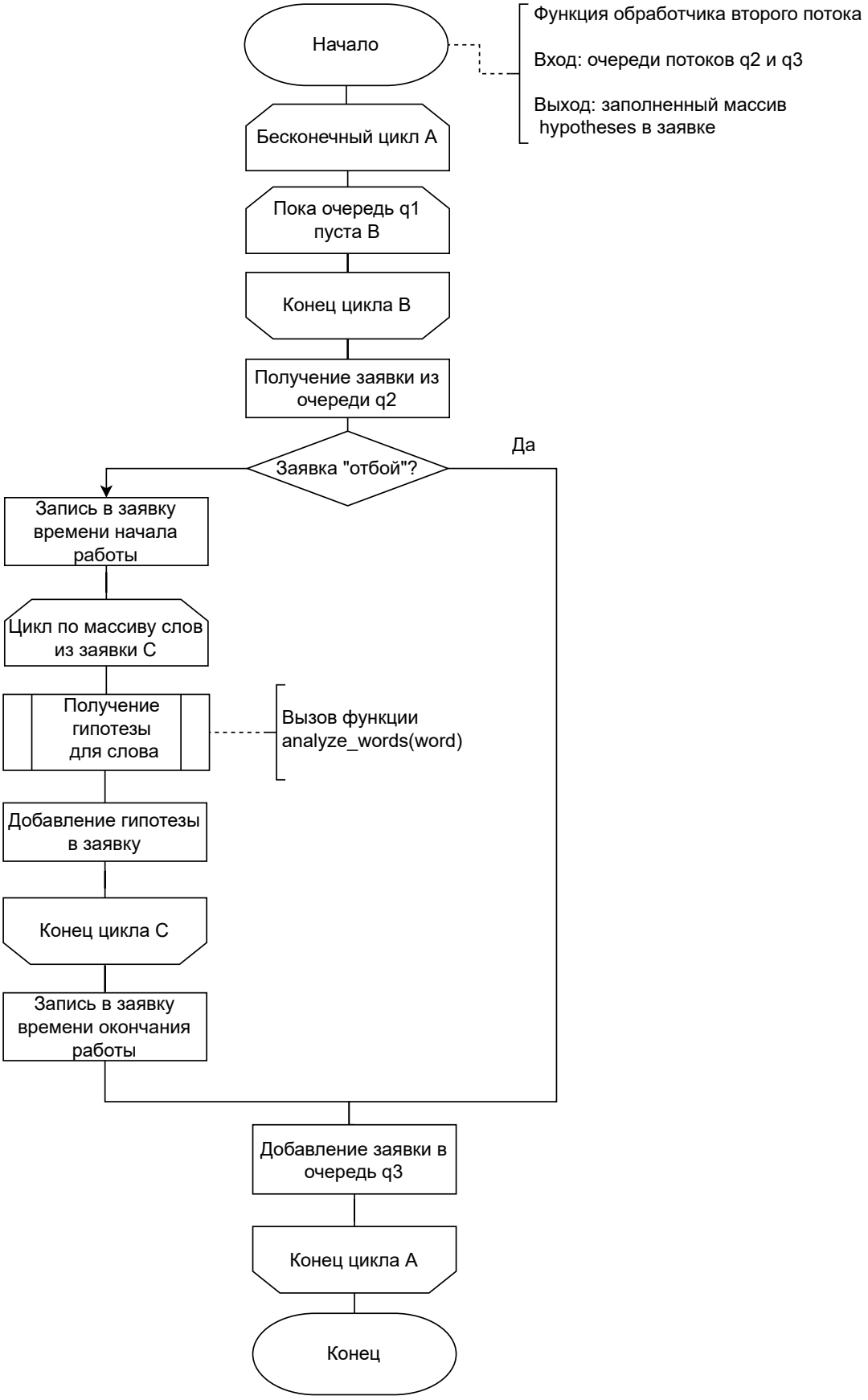

Рисунок 2.3 – Схема алгоритма потока 2

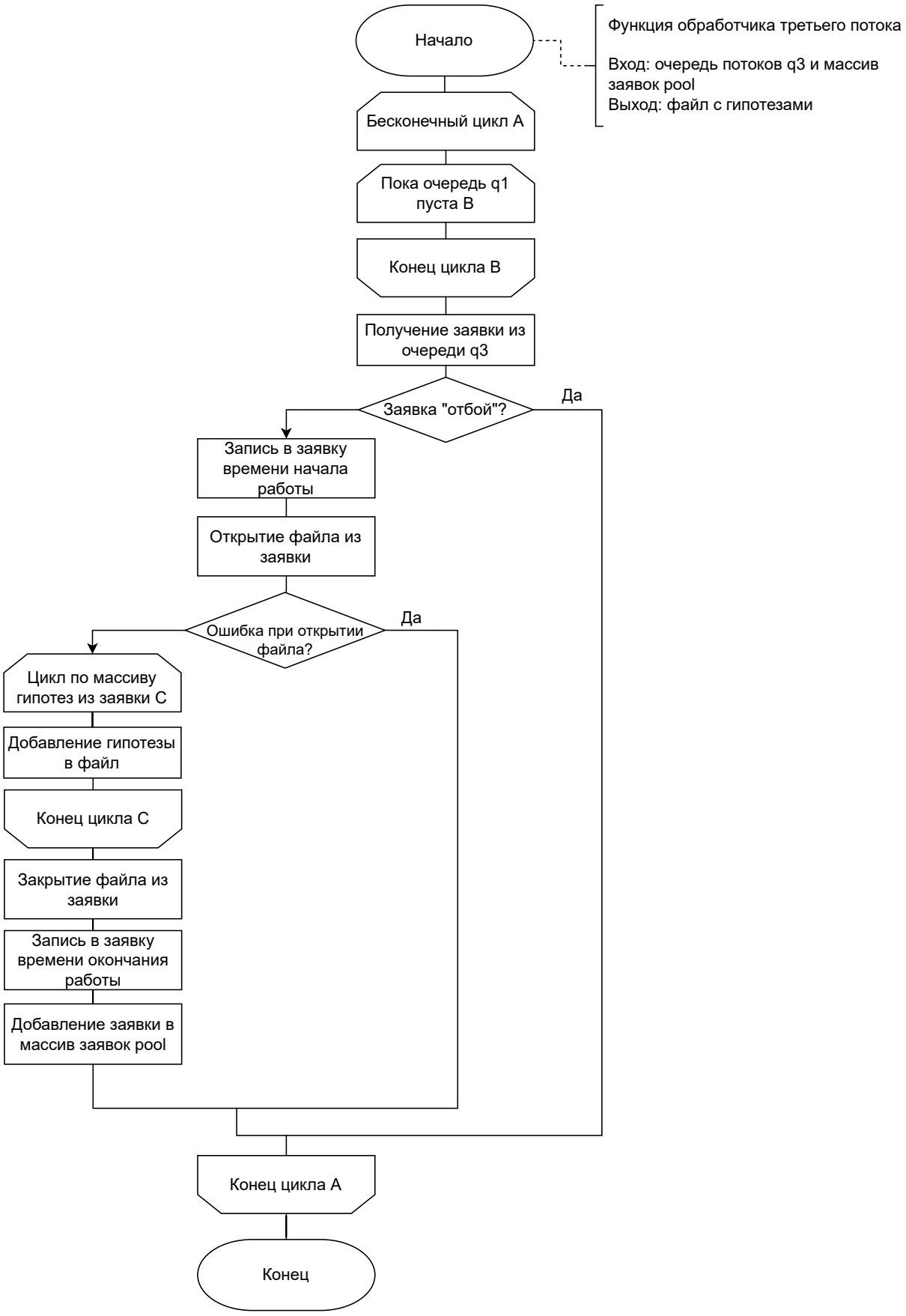

Рисунок 2.4 – Схема алгоритма потока 3

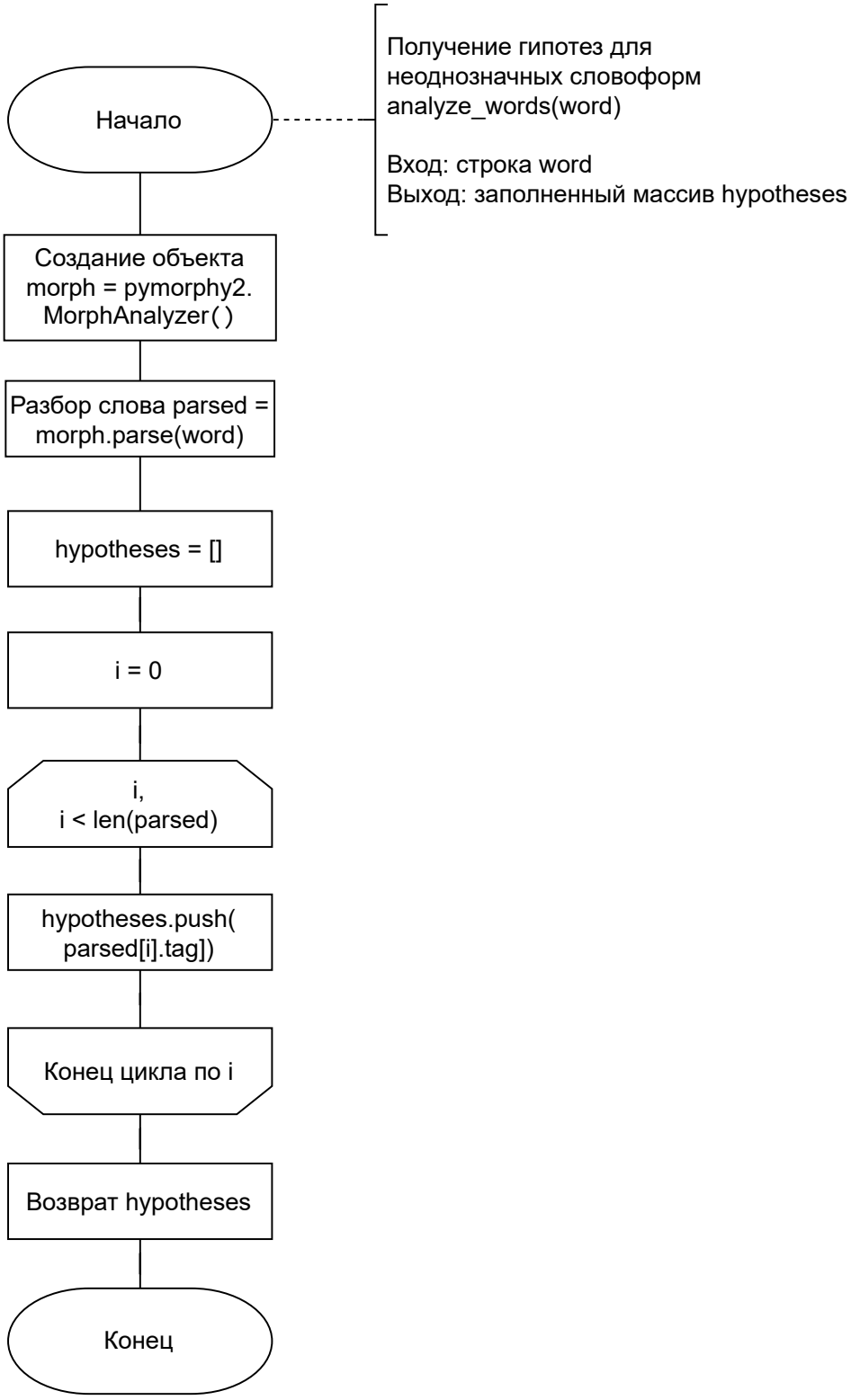

Рисунок 2.5 – Схема алгоритма получения гипотез для неоднозначных словоформ

Вывод

В данном разделе на основе теоретических данных были построены схемы требуемых алгоритмов.

3 Технологическая часть

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

3.1 Требования к программному обеспечению

К программе предъявлен ряд требований:

— принимает на вход файл с неоднозначными словоформами;

— создает и записывает в файл результат работы алгоритма;

— имеет интерфейс для выбора действий;

— выводит лог работы конвейера;

— имеет функциональность замера времени работы реализации алгоритма.

3.2 Средства реализации

Для реализации данной лабораторной работы был выбран язык C++ и Python [4],[5]. Данный выбор обусловлен наличием у языка C++ всех требующихся инструментов для организации конвейера в лабораторной работе таких как:

— работа с потоками осуществляется классом *thread* [6];

— работа с мьютексами осуществляется классом *mutex* [7];

— работа с очередями осуществляется классом *queue* [8].

Выбор языка Python обусловлен возможностью работать с библиотекой *pymorphy2*.

Время выполнения программы было замерено с использованием функции *high_resolution_clock* из библиотеки *std* :: *chrono* [9].

3.3 Сведения о модулях программы

Данная программа разбита на следующие модули:

— `main.cpp` — файл, содержащий точку входа в программу, из которой происходит запуск работы реализации алгоритма.

— `pymorphy.py` — файл, содержащий функции для работы с библиотекой *pymorphy*2.

— `graph.py` — файл, содержащий функции для построения графиков.

3.4 Реализация алгоритмов

В листингах 3.1 – 3.6 приведены реализация конвейерной обработки и реализация каждой стадии в отдельном вспомогательном потоке. В листинге 3.7 приведен алгоритм получения гипотез для неоднозначных словоформ.

Листинг 3.1 – Функция реализации основного потока

```
int pipeline(int number_of_files, std::string file_path) {
    std::queue<Request> inputQueue;
    std::queue<Request> pythonQueue;
    std::queue<Request> outputQueue;
    std::vector<Request> pool(number_of_files);

    for (int i = 0; i < number_of_files; i++) {
        Request request;
        request.id = i;
        request.file_path = file_path;
        inputQueue.push(request);
    }

    Request terminationRequest;
    terminationRequest.id = -1;
    inputQueue.push(terminationRequest);

    std::thread t1(handler1, std::ref(inputQueue),
       std::ref(pythonQueue));
    std::thread t2(handler2, std::ref(pythonQueue),
       std::ref(outputQueue));
    std::thread t3(handler3, std::ref(outputQueue),
       std::ref(pool));

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

    createLog(pool);

    return 0;
}
```

Листинг 3.2 – Функция реализации первого потока для считывания слов в файл

```
1  void handler1(std::queue<Request>& inputQueue,
       std::queue<Request>& outputQueue) {
2      while (true) {
3
4          while (inputQueue.empty());
5
6          Request request = inputQueue.front();
7          inputQueue.pop();
8
9          if (request.id == -1) {
10             outputQueue.push(request);
11             break;
12         }
13
14         request.t1_start =
               std::chrono::high_resolution_clock::now();
15
16         std::ifstream file(request.file_path);
17         if (!file.is_open()) {
18             exit(EXIT_FAILURE);
19         }
20
21         std::string word;
22         while (file >> word) {
23             request.words.push_back(word);
24         }
25
26         file.close();
27
28         request.t1_end =
               std::chrono::high_resolution_clock::now();
29         std::unique_lock<std::mutex> lock2(mutex2);
30         outputQueue.push(request);
31         lock2.unlock();
32     }
33 }
```

Листинг 3.3 – Функция реализации второго потока для вызова функции получения гипотез

```
void handler2(std::queue<Request>& inputQueue,
    std::queue<Request>& outputQueue) {
    while (true) {
        while (inputQueue.empty());
        std::unique_lock<std::mutex> lock2(mutex2);

        Request request = inputQueue.front();
        inputQueue.pop();

        lock2.unlock();

        if (request.id == -1) {
            outputQueue.push(request);
            break;
        }

        request.t2_start =
            std::chrono::high_resolution_clock::now();

        for (const auto& word : request.words) {
            std::string hypothes = pymorphy(word.c_str(),
                "pymorphy.py");
            request.hypotheses.push_back(hypothes);
        }

        request.t2_end =
            std::chrono::high_resolution_clock::now();

        std::unique_lock<std::mutex> lock3(mutex3);
        outputQueue.push(request);
        lock3.unlock();
    }
}
```

Листинг 3.4 – Функция для вызова скрипта получения гипотез

```
std::string pymorphy(const char* inputWord, const char*
    pyScript) {
    char line[100];
    sprintf(line, "python3 %s %s", pyScript, inputWord);

    FILE* pipe = popen(line, "r");
    if (!pipe) {
        std::cerr << "Ошибка с вызовом python" << std::endl;
        exit(EXIT_FAILURE);
    }

    char res[1000];
    fgets(res, sizeof(res), pipe);
    pclose(pipe);

    return std::string(res);
}
```

Листинг 3.5 – Функция реализации третьего потока для записи гипотез в файл

```
while (true) {
    while (inputQueue.empty());

    std::unique_lock<std::mutex> lock3(mutex3);
    Request request = inputQueue.front();

    inputQueue.pop();
    lock3.unlock();

    if (request.id == -1) break;

    request.t3_start =
       std::chrono::high_resolution_clock::now();

    std::string output_file_path = "./output" +
       std::to_string(request.id) + ".txt";
    std::ofstream output_file(output_file_path);

    if (!output_file.is_open()) {
        exit(EXIT_FAILURE);
    }

    for (const auto& hypothes : request.hypotheses) {
        output_file << hypothes << std::endl;
    }
    output_file.close();
    request.t3_end = std::chrono::high_resolution_clock::now();
    pool[request.id] = request;
}
```

Листинг 3.6 – Функция вывода лога программы

```
1    void createLog(const std::vector<Request>& pool) {
2        std::vector<Log> logs;
3        for (const auto& request : pool) {
4            logs.push_back({1, request.id, 1,
                 request.t1_start});
5            logs.push_back({1, request.id, 0, request.t1_end});
6            logs.push_back({2, request.id, 1,
                 request.t2_start});
7            logs.push_back({2, request.id, 0, request.t2_end});
8            logs.push_back({3, request.id, 1,
                 request.t3_start});
9            logs.push_back({3, request.id, 0, request.t3_end});
10       }
11
12       std::sort(logs.begin(), logs.end());
13
14       for (const auto& log : logs) {
15           std::string message = log.type ? " начал" : " законч
                 ил";
16           std::cout << log.id << message << " работать с " <<
                 log.request_id << " заявкой " << "("<<
                 std::chrono::duration_cast
                 <std::chrono::nanoseconds>(
                 log.time.time_since_epoch()).count() <<")" <<
                 std::endl;
17       }
18   }
```

Листинг 3.7 – Функция алгоритма получения гипотез для неоднозначных словоформ

```
def analyze_word(word):
    morph = pymorphy2.MorphAnalyzer()
    parsed = morph.parse(word)
    hypotheses = []
    for p in parsed:
        hypotheses.append(p.tag)

    return hypotheses

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Неверный формат запуска")
        sys.exit(1)
    word = sys.argv[1]
    hypotheses = analyze_word(word)
    print(hypotheses)
```

3.5 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для вышеизложенных алгоритмов. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Словоформа	**Результат**
'атлас'	('NOUN,inan,masc sing,nomn')
'запись'	('NOUN,inan,femn sing,nomn')
'тир'	('NOUN,inan,masc sing,accs')

Вывод

В данном разделе были рассмотрены средства реализации, был представлен листинг реализаций алгоритма получения гипотез для неоднозначных словоформ. Также было проведено тестирование реализаций алгоритма.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программ, проведение исследования и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени:

— Процессор: 11th Gen Intel(R) Core(TM) i5-1135G7 2.42 ГГц.

— Количество ядер: 4 физических и 8 логических ядер.

— Оперативная память: 16 ГБайт.

— Операционная система: Windows 11 Домашняя 64-разрядная система версии 22H2 [10].

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

4.2 Демонстрация работы программы

Программа получает на вход номер режима. На выходе составляется файл с гипотезами и выводится лог программы.

На рисунке 4.1 представлена демонстрация работы программы.

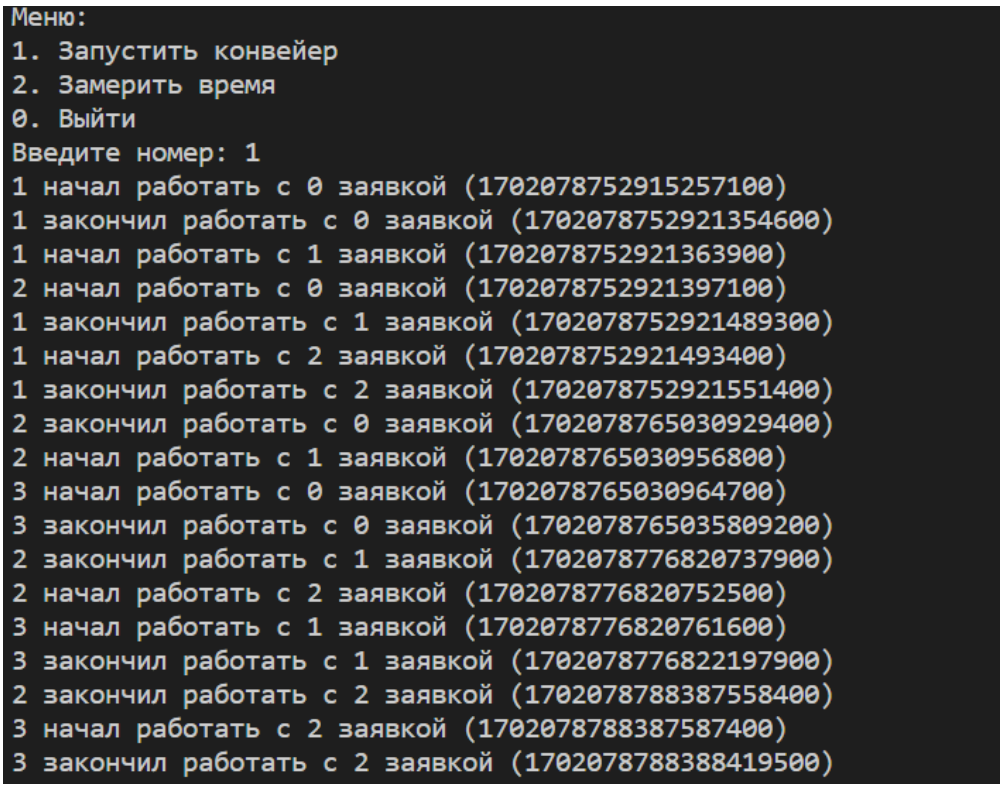

Рисунок 4.1 – Демонстрация интерфейса программы

4.3 Временные характеристики

Результаты эксперимента замеров по времени приведены в таблице 4.1. Замеры проводились для набора значений заявок от 1 до 7 с шагом 1.

Таблица 4.1 – Результаты нагрузочного тестирования

Количество заявок	Время (с)
1	11.1764
2	21.8164
3	32.6525
4	43.544
5	54.6064
6	65.5844
7	76.9687

По таблице 4.1 был построен график 4.2. Из полученных результатов можно сделать вывод, что зависимость количества заявок от времени выполнения линейна.

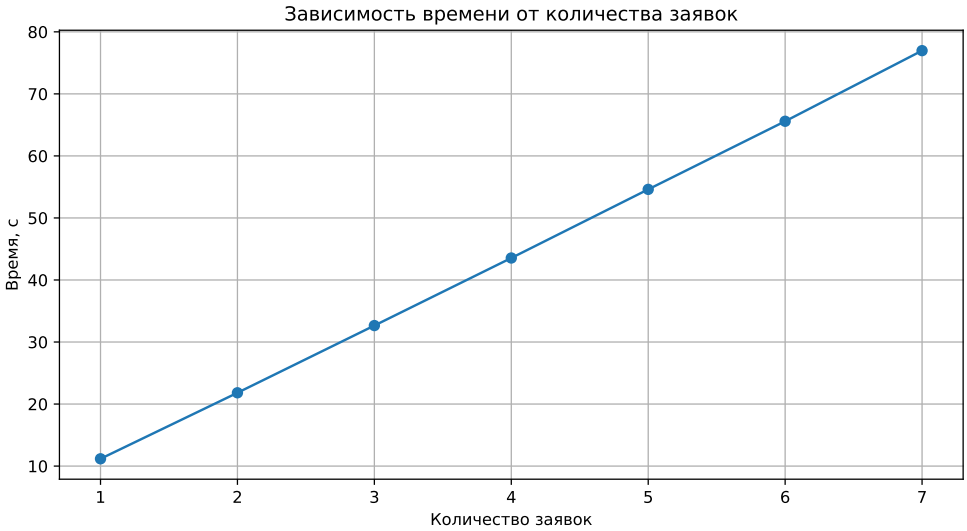

Рисунок 4.2 – График зависимости времени от количества заявок

Также была построена таблица замеров времени пребывания заявки в системе 4.2. В результате было выяснено, что дольше всего заявка обрабатывается во втором потоке, где реализована работа с *pymorphy*2 через Python. Обозначения в таблице:

— count — количество заявок;

— tmin — минимальное время пребывания в системе;

— tavg — среднее время пребывания в системе;

— tmax — максимальное время пребывания в системе;

— t2 — время в потоке 2;

— t3 — время в потоке 3;

— all — время в системе.

Таблица 4.2 – Результаты замера времени пребывания заявки (мс)

count	tmin	tavg	tmax	t2	t3	all
1	0	3829.33	11488	11488	0	11489
2	0	4379.5	13338	13138	0.5	19808.5
3	0	4426.44	13383	13279	0.333333	26508.7
4	0	4323	13432	12960.8	8.25	32652.2
5	0	4226.73	13150	12673.6	6.6	38146.6

4.4 Вывод

В данном разделе было произведен анализ реализации алгоритма по затраченному времени.

В результате эксперимента было выявлено, что зависимость количества заявок от времени выполнения линейна. Также было проанализировано полное время заявки в системе и в результате выяснено, что дольше всего заявка обрабатывается во втором потоке, где реализована работа с *pymorphy2* через Python.

Заключение

В ходе выполнения лабораторной работы было выявлено, зависимость количества заявок от времени выполнения линейна. Также было проанализировано полное время заявки в системе и в результате выяснено, что дольше всего заявка обрабатывается во втором потоке, где реализована работа с *pymorphy*2 через Python.

Цель, которая была поставлена в начале лабораторной работы была достигнута, а также в ходе выполнения лабораторной работы были решены следующие задачи.

1) Описана организация конвейерной обработки данных.

2) Описаны алгоритмы обработки данных, которые были использованы в лабораторной работе.

3) Реализована программа, выполняющая конвейерную обработку с количеством лент не менее трех.

4) Проанализирована реализация алгоритма по затраченному времени.

Список использованных источников

1. Конвейерная обработка данных [Электронный ресурс]. — Режим доступа: https://studref.com/636041/ekonomika/konveyernaya_obrabotka_dannyh (дата обращения: 5.12.2023).

2. Документация по pymorphy2 [Электронный ресурс]. — Режим доступа: https://pymorphy2.readthedocs.io/en/stable/index.html (дата обращения: 5.12.2023).

3. У. Ричард Стивенс Стивен А. Раго. UNIX. Профессиональное программирование. 3-е издание. — СПб.: Питер, 2018. С. 649–664.

4. C++ language [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/ (дата обращения: 5.12.2023).

5. Документация по python [Электронный ресурс]. — Режим доступа: https://docs.python.org/3/ (дата обращения: 5.12.2023).

6. Класс thread [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/ru-ru/cpp/standard-library/thread-class (дата обращения: 5.12.2023).

7. Класс mutex [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/ru-ru/cpp/standard-library/mutex-class-stl (дата обращения: 5.12.2023).

8. Класс queue [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/ru-ru/cpp/standard-library/queue-class (дата обращения: 5.12.2023).

9. *clock_gettime* function [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/chrono (дата обращения: 6.12.2023).

10. Windows 11 [Электронный ресурс]. — Режим доступа: https://www.microsoft.com/ru-ru/software-download/windows11 (дата обращения: 12.11.2023).