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) Разин А. (И. О. Фамилия)

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

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Разработчики архитектуры компьютеров издавна прибегали к методам проектирования, известным под общим названием «совмещение операций», при котором аппаратура компьютера в любой момент времени выполняет одновременно более одной базовой операции.

Этот метод включает в себя, в частности, такое понятие, как конвейеризация. Конвейры широко применяются программистами для решения трудоемких задач, которые можно разделить на этапы, а также в большинстве современных быстродействующих процессоров [1].

Целью данной работы является получение навыков организации конвейерной обработки данных.

В рамках выполнения работы необходимо решить следующие задачи:

1. описать организацию конвейерной обработки данных;
2. описать алгоритмы обработки данных, которые будут использоваться в текущей лабораторной работе;
3. реализовать программу, выполняющую конвейерную обработку с количеством лент не менее трех в однопоточной и многопоточной реализаций;
4. провести сравнительный анализ времени работы реализаций.

1 Аналитическая часть

В данной части работы будут рассмотрены основы конвейерной обработки данных, а также будет описан алгоритм сортировки слиянием.

1.1 Конвейерная обработка данных

Конвейеризация (или конвейерная обработка) в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями, 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 [2].

1.2 Сортировка слиянием

Алгоритм сортировки слиянием (англ. merge sort) является эффективным и стабильным алгоритмом сортировки, который применяет принцип «разделяй и властвуй» для упорядочивание элементов в массиве [3].

Алгоритм состоит из нескольких этапов [3].

1. Разделение — исходный массив разделяется на две равные (или почти равные) половины. Это делается путем нахождения середины массива и создания двух новых массивов, в которые будут скопированы элементы из левой и правой половин.

2. Рекурсивная сортировка: каждая половина массива рекурсивно сортируется с помощью алгоритма сортировки слиянием. Этот шаг повторяется до тех пор, пока размер каждой половины не станет равным 1.

3. Слияние: отсортированные половины массива объединяются в один отсортированный массив. Для этого создается новый массив, в который будут последовательно добавляться элементы из левой и правой половин. При добавлении элементов выбирается наименьший элемент из двух половин и добавляется в новый массив. Этот шаг повторяется до тех пор, пока все элементы не будут добавлены в новый массив.

В результате выполнения этих шагов получается отсортированный массив, который содержит все элементы исходного массива.

1.3 Операции на конвейере

В качестве операций, выполняющихся на конвейере, взяты следующие:

1. сортировка слиянием копии массива, предоставленного в заявке;
2. сортировка слиянием с использованием 4 дополнительных потоков исходного массива;
3. запись значений отсортированного массива в файл, соответствующий номеру заявки.

Вывод

В данной части была рассмотрена конвейерная обработка данных и описан алгоритм сортировки слиянием, а также описаны конкретные операции на конвейере.

2 Конструкторская часть

В данной части работы будут рассмотрены схемы алгоритмов сортировок, а также схемы параллельной и последовательной обработкой конвейра.

2.1 Схемы алгоритмов

На рисунках 2.1–2.5 приведены схемы алгоритмов линейной обработки заявок и конвейерной обработки заявок, также представлена схема каждого этапа обработки заявок.

Вывод

В данном разделе на основе теоретических данных были построены схемы требуемых алгоритмов, а также для каждого алгоритма сортировки были выведены трудоемкости худшего и лучшего случаев.

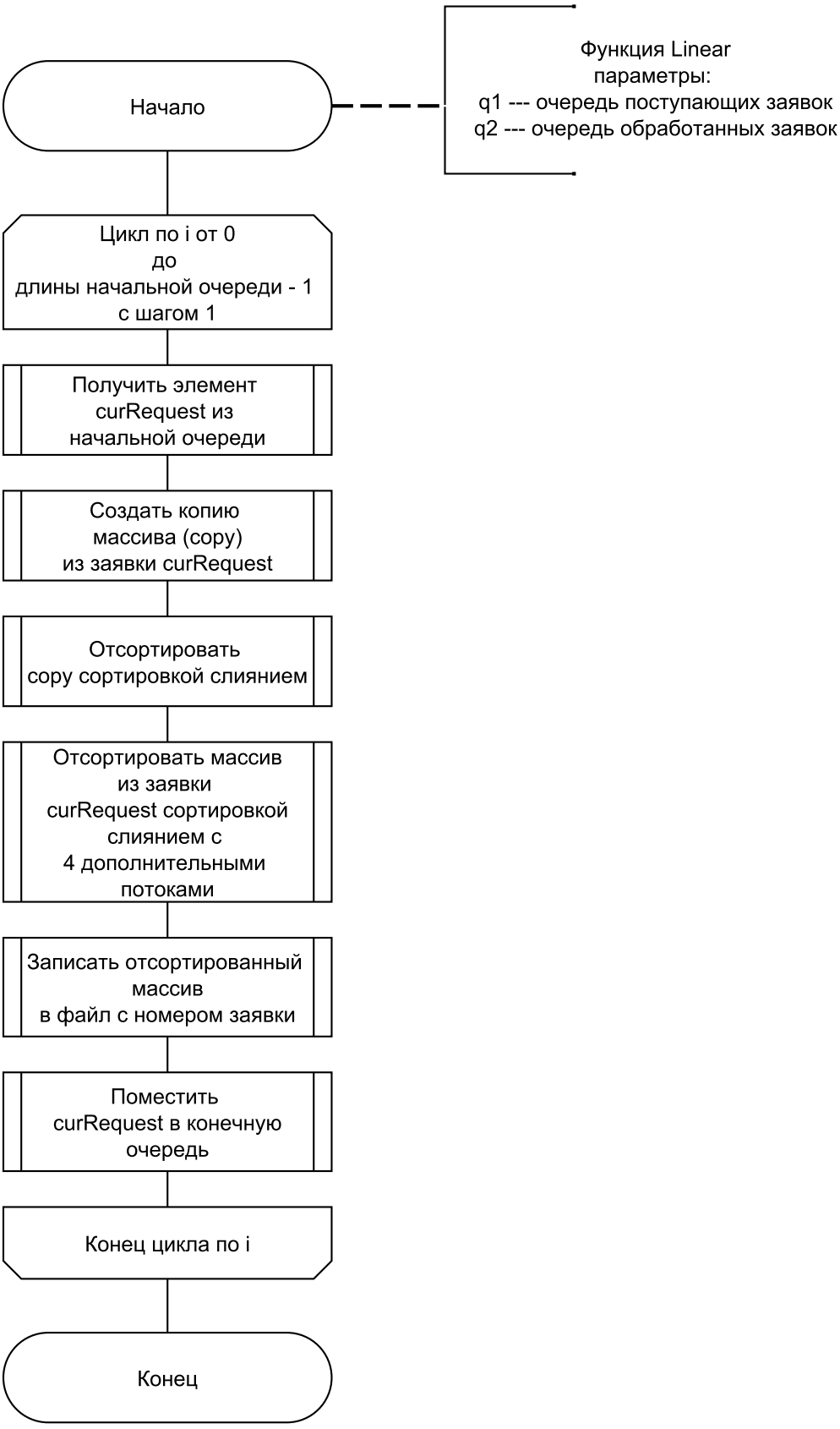

Рисунок 2.1 – Схема линейного алгоритма обработки заявок

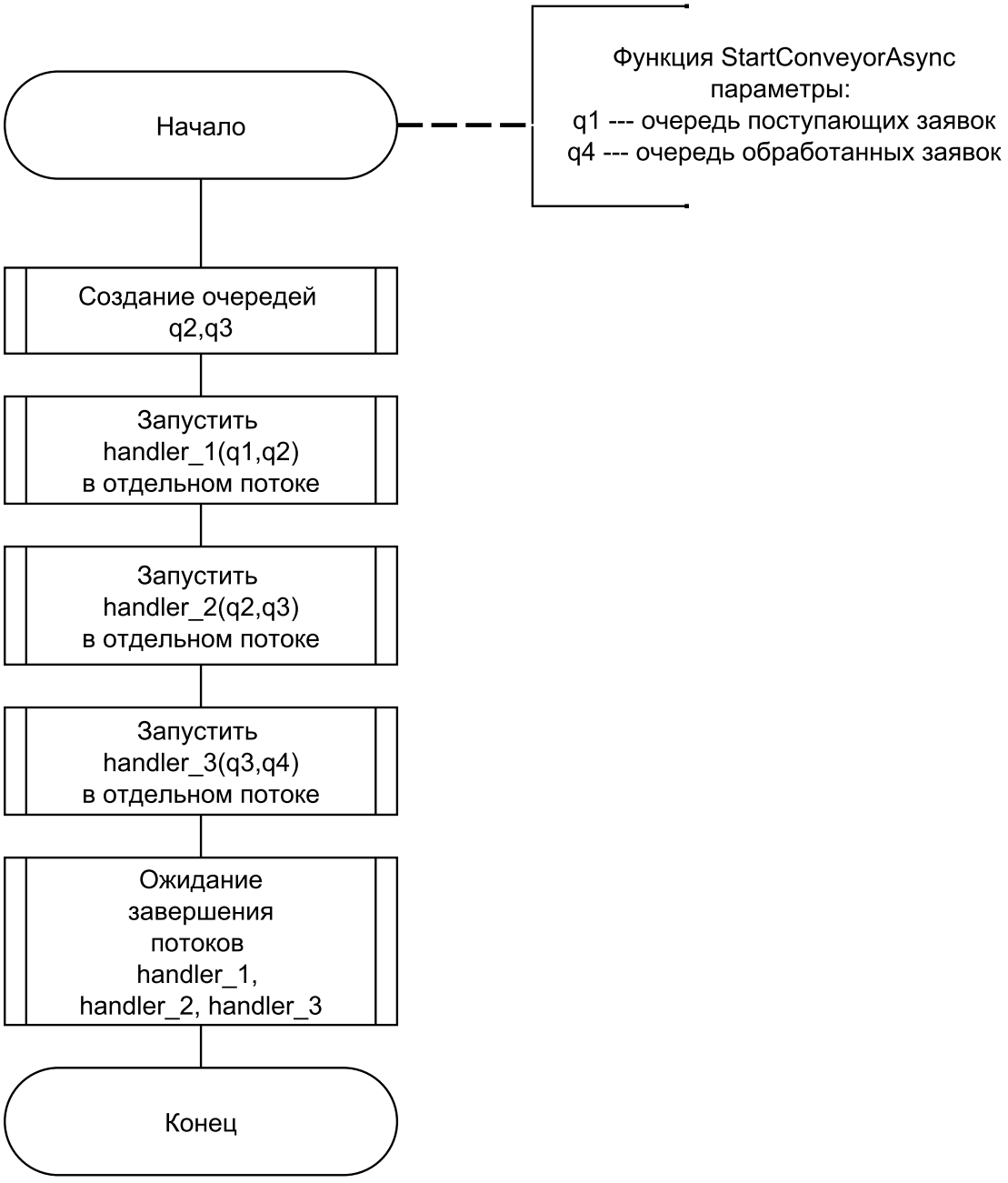

Рисунок 2.2 – Схема алгоритма конвейерной обработки заявок

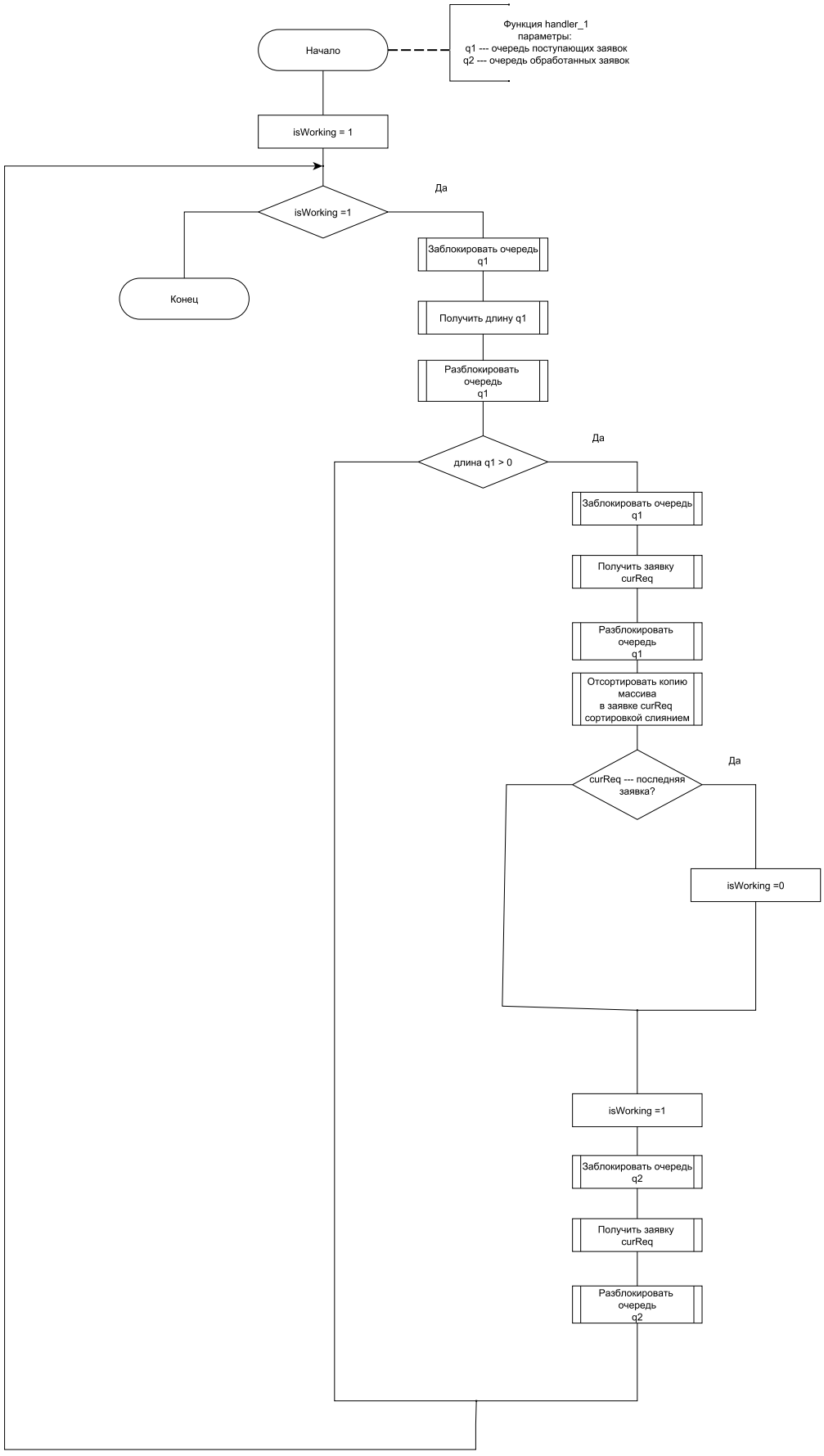

Рисунок 2.3 – Схема алгоритма 1 этапа обработки заявки (сортировка копии массива)

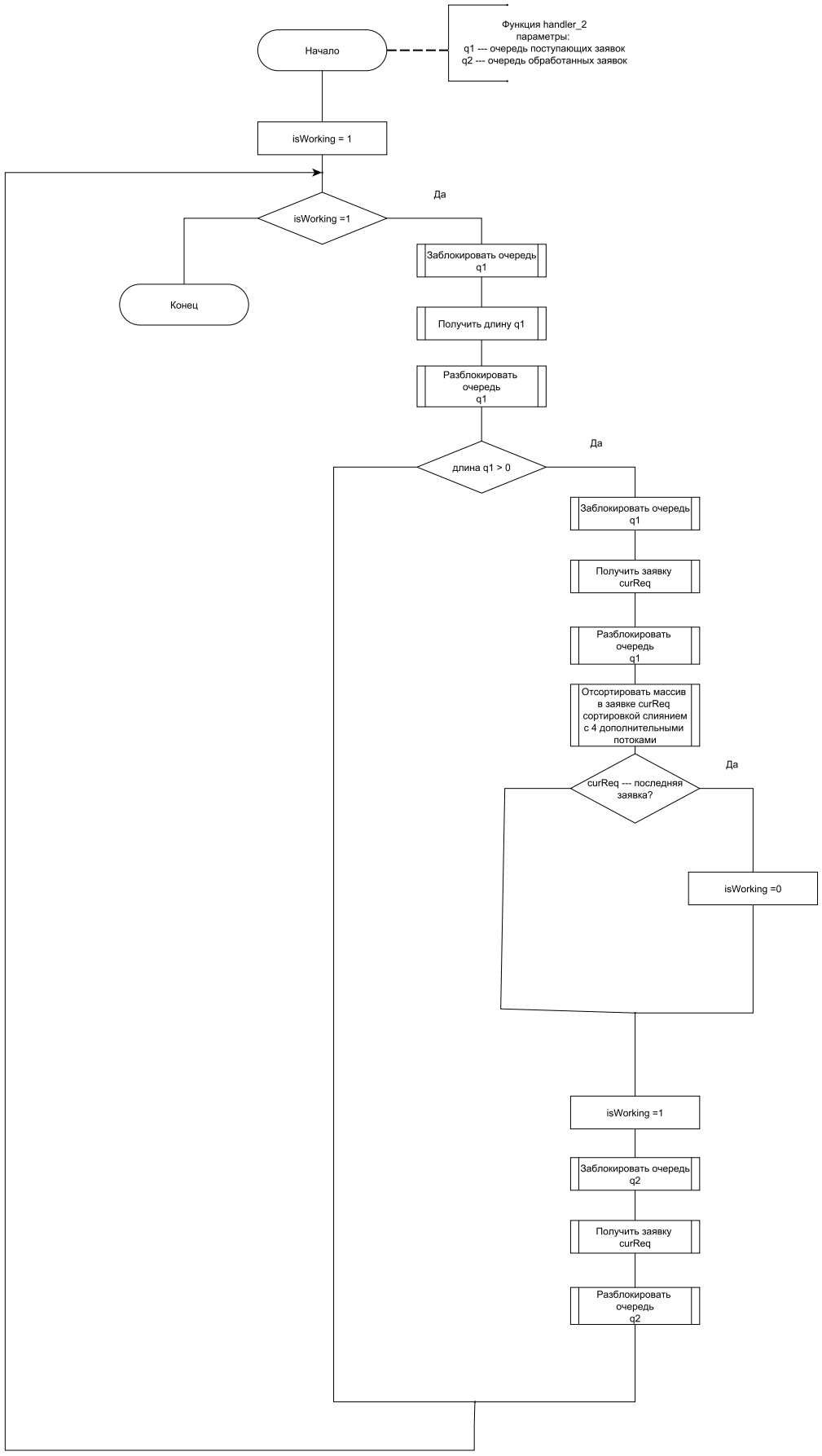

Рисунок 2.4 – Схема алгоритма 2 этапа обработки заявки (сортировка массива с использованием дополнительных потоков)

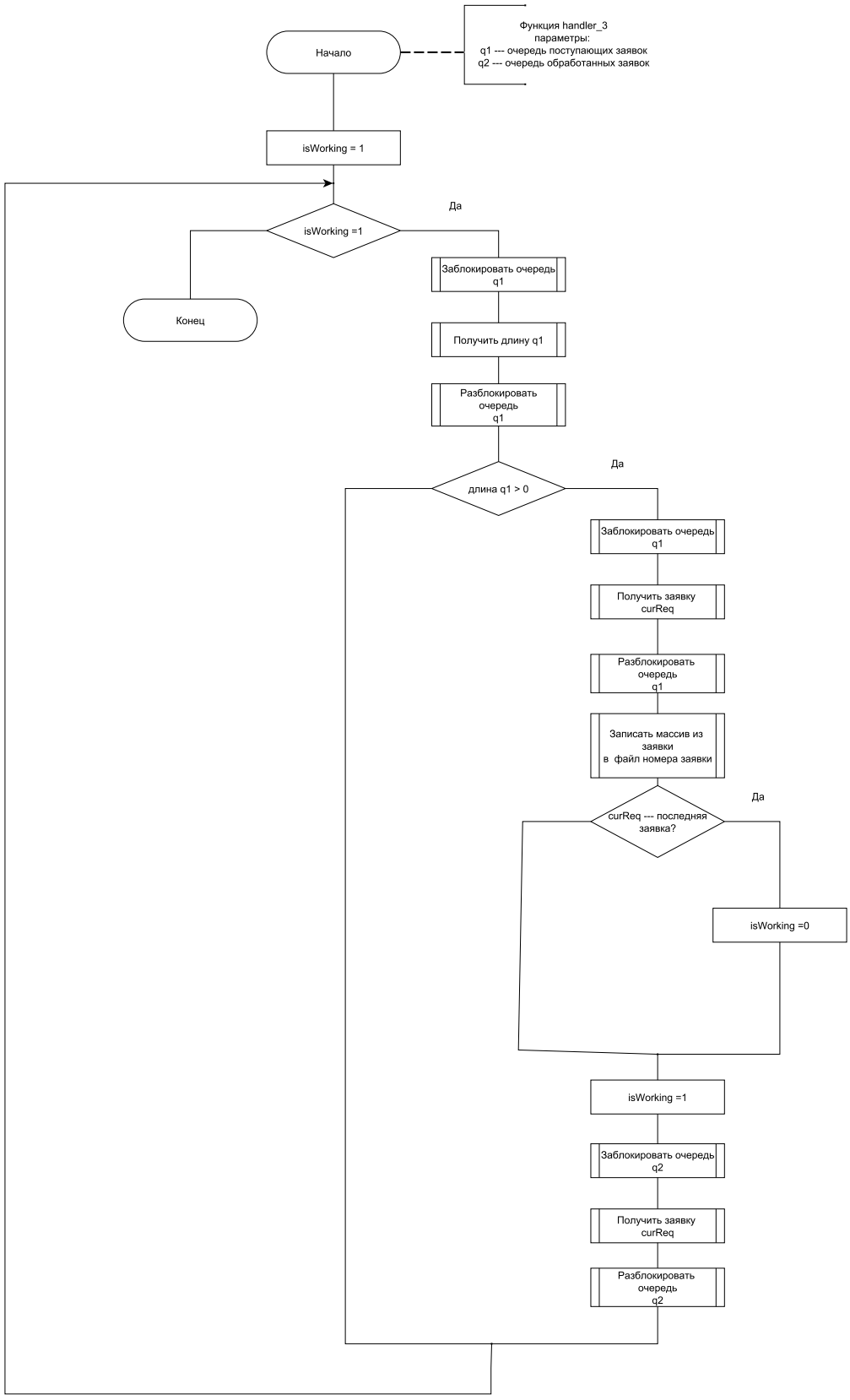

Рисунок 2.5 – Схема алгоритма 3 этапа обработки заявки (запись массива в файл)

3 Технологический раздел

В данной части работы будут описаны средства реализации программы, а также листинги, модульные и функциональные тесты.

3.1 Средства реализации

Алгоритмы для данной лабораторной работы были реализованы на языке C++, при использовании компилятора gcc версии 10.5.0, так как в стандартной библиотеке приведенного языка присутствует функция `clock_gettime`, которая (при использовании макропеременной `CLOCK_THREAD_CPUTIME_ID`) позволяет получить процессорное время, затрачиваемое конкретным потоком [4].

3.2 Реализация алгоритмов

Листинги исходных кодов программ А.1–А.5 приведены в приложении.

3.3 Тестирование

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

Таблица 3.1 – Модульные тесты

Массив	**Размер**	**Ожидаемый р-т**	**Фактический результат**	
			Блочная/Перемеш.	**Поразрядная**
1 2 3 4	4	1 2 3 4	1 2 3 4	1 2 3 4
4 3 2 1	4	4 3 2 1	4 3 2 1	4 3 2 1
3 5 1 6	4	1 3 5 6	1 3 5 6	1 3 5 6
-5 -1 -3 -4 -2	5	-5 -4 -3 -2 -1	-5 -4 -3 -2 -1	-5 -4 -3 -2 -1
1 -3 2 9 -9	5	-9 -3 1 2 9	-9 -3 1 2 91	-9 -3 1 2 9

Вывод

В данной части работы были представлены листинги реализованных алгоритмов и тесты, успешно пройденные программой.

4 Исследовательская часть

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

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени, представлены далее.

1. Процессор Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 2592 МГц, ядер: 6, логических процессоров: 12.
2. Оперативная память: 16 ГБайт.
3. Операционная система: Майкрософт Windows 10 Pro [5].
4. Использованная подсистема: WSL2 [6].

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

4.2 Демонстрация работы программы

На рисунке 4.1 представлена демонстрация работы разработанного приложения для алгоритмов сортировок.

```
Меню:
0)Выход
1)Сортировка массива с использованием сортировки перемешивания
2)Сортировка массива с использованием поразрядной сортировки
3)Сортировка массива с использованием блочной сортировки
4)Расчет времени

Введите пункт из меню: 1

Введите  размер массива для сортировки:
4
Введите массив для сортировки:
4 2 1 3
Отсортированный массив
1 2 3 4
```

Рисунок 4.1 – Пример работы программы

4.3 Временные характеристики

Результаты исследования замеров по времени приведены в таблице 4.1.

Таблица 4.1 – Полученная таблица замеров по времени различных реализаций алгоритмов сортировки

n	Поразрядная(мс)	Блочная(мс)	Перемешиванием(мс)
1000	0.15692	1.2194	2.03
2000	0.31811	4.5549	8.2541
3000	0.47956	10.05	18.767
4000	0.62436	17.109	32.384
5000	0.78017	26.401	50.514
6000	0.93926	37.91	72.854
7000	1.0944	51.295	99.256
8000	1.2494	66.616	129.05
9000	1.6587	97.898	191.14
10000	1.5698	104.19	203.08

Для таблицы 4.1 расчеты проводились с шагом 1000, сортировки производились 1000 раз, после чего результат усредняется. В качестве сортируемых значений использовались числа от 1 до 10000000. Размерность блоков определялась как $\frac{n}{2} + 2$. Данные генерировались из равномерного распределения.

По таблице 4.1 были построены графики ?? – ??.

В результате анализа таблицы 4.1, было получено, что при 10000 элементов реализация алгоритма блочной сортировки требует в 1.95 раз больше времени, чем сортировка перемешиванием, поразрядная сортировка при 10000 элементах требует в 65 раз меньше времени, чем блочная сортировка . Ввиду того, что сортируемые числа неотрицательные и имеют малое число разрядов поразрядная сортировка оказалась самой эффективной по временным затратам, блочная сортировка показала лучший результат по сравнению с сортировкой перемешиваем благодаря равномерному распределению данных.

ЗАКЛЮЧЕНИЕ

В результате исследования было определено, что реализация алгоритма блочной сортировки при 10000 элементах требует в 1.95 раз больше времени, чем сортировка перемешиванием, поразрядная сортировка при 10000 элементах требует в 65 раз меньше времени, чем блочная сортировка. Ввиду того, что сортируемые числа неотрицательные и имеют малое число разрядов поразрядная сортировка оказалась самой эффективной по временным затратам, блочная сортировка показала лучший результат по сравнению с сортировкой перемешиваем благодаря равномерному распределению данных.

Поставленная цель: описание и исследование трудоемкости алгоритмов сортировки, была достигнута.

Для поставленной цели были выполнены все задачи.

1. Описать алгоритмы сортировки.
2. Создать программное обеспечение, реализующее следующие алгоритмы сортировки:
   - перемешиванием;
   - блочная;
   - поразрядная.
3. Оценить трудоемкости сортировок.
4. Замерить время реализации.
5. Провести анализ затрат работы программы по времени, выяснить влияющие на них характеристики.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Принципы конвейерной технологии [Электронный ресурс]. — Режим доступа: https://www.sites.google.com/site/shoradimon/18-principy-konvejernoj-tehnologii (дата обращения: 24.12.2023).
2. Конвейерная обработка данных [Электронный ресурс]. — Режим доступа: https://studfile.net/preview/1083252/page:25/ (дата обращения: 24.12.2023).
3. Merge sort [Электронный ресурс]. — Режим доступа: https://nauchniestati.ru/spravka/algoritm-sortirovki-sliyaniem (дата обращения: 06.12.2023).
4. C library function clock() [Электронный ресурс]. — — Режим доступа: https://linux.die.net/man/3/clock_gettime (дата обращения: 28.09.2023).
5. Windows 10 Pro 2h21 64-bit [Электронный ресурс]. — — Режим доступа: https://www.microsoft.com/ru-ru/software-download/windows10 (дата обращения: 28.09.2023).
6. Что такое WSL [Электронный ресурс]. — — Режим доступа: https://learn.microsoft.com/ru-ru/windows/wsl/about (дата обращения: 28.09.2023).

ПРИЛОЖЕНИЕ А

Листинг А.1 – Реализация алгоритма поразрядной сортировки

```
void radixSort(std::vector<int>& arr)
{

    int m = getMaxAbs(arr);
    for (int exp = 1; m / exp > 0; exp *= 10)
    countSort(arr, exp);
}
```

Листинг А.2 – Сортировка подсчетом разрядов чисел

```
void countSort(std::vector<int>& arr, int exp)
{

    int n = arr.size();
    int output[n];
    int count[20] = { 0 };
    int i;

    for (i = 0; i < n; i++)
    count[(arr[i] / exp) \% 10 + 9]++;

    for (i = 1; i < 20; i++)
    count[i] += count[i - 1];

    for (i = n - 1; i >= 0; i--)
    {
        output[count[(arr[i] / exp) \% 10 + 9] - 1] = arr[i];
        count[(arr[i] / exp) \% 10 + 9]--;
    }

    for (i = 0; i < n; i++)
    arr[i] = output[i];
}
```

Листинг А.3 – Функция поиска максимального значения по модулю в векторе

```
int getMaxAbs(const std::vector<int>& arr)
{
```

```
    int mx = abs(arr[0]);
    for (size_t i = 1; i < arr.size(); i++)
    if (arr[i] > abs(mx))
    mx = arr[i];
    return mx;
}
```

Листинг А.4 – Реализация алгоритма сортировки перемешиванием

```
void shakerSort(std::vector<int>& arr)
{
    int control = arr.size() - 1;
    int left = 0, right = control;
    do
    {
        for (int i = left; i < right; i++)
        {
            if (arr[i] > arr[i + 1])
            {
                std::swap(arr[i], arr[i + 1]);
                control = i;
            }
        }
        right = control;
        for (int i = right; i > left; i--)
        {
            if (arr[i] < arr[i - 1])
            {
                std::swap(arr[i], arr[i - 1]);
                control = i;
            }
        }
        left = control;
    } while (left < right);
}
```

Листинг А.5 – Реализация алгоритма блочной сортировки

```
    void blockSort(std::vector<int>& arr, int blockSize)
    {
        std::vector<std::vector<int> > blocks;

        for (size_t i = 0; i < arr.size(); i += blockSize)
        {
```

```
            std::vector<int> block;

            for (size_t j = i; j < i + blockSize && j <
               arr.size();
            j++)
            {
                block.push_back(arr[j]);
            }

            shakerSort(block);
            blocks.push_back(block);
        }

        int arrIndex = 0;
        while (!blocks.empty())
        {

            int minIdx = 0;
            for (size_t i = 1; i < blocks.size(); i++)
            {
                if (blocks[i][0] < blocks[minIdx][0])
                {
                    minIdx = i;
                }
            }

            arr[arrIndex] = blocks[minIdx][0];
            arrIndex++;
            blocks[minIdx].erase(blocks[minIdx].begin());

            if (blocks[minIdx].empty())
            {
                blocks.erase(blocks.begin() + minIdx);
            }
        }

    }
```