Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский институт)»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

О т ч ё т

п о л а б о р а т о р н о й р а б о т е 5

Дисциплина: Анализ Алгоритмов

Тема лабораторной работы работы:
Конвейерная обработка данных

Студентки гр. ИУ7-51б _____ **Сушина А.Д.**

Преподаватель _____ **Волкова Л.Л.**

Москва, 2019г

Оглавление

Введение

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.

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

Для достижения поставленной цели требуется выполнить следующие **задачи.**

1. Выбрать и описать методы обработки данных, которые будут сопоставлены методам конвейера.
2. Описать архитектуру программы, а именно какие функции имеет главный поток, принципы и алгоритмы обмена данными между потоками.
3. Реализовать конвейерную систему, а также сформировать лог событий с указанием времени их происхождения, описать реализацию.
4. Провести тестирование системы.
5. Интерпретировать сформированный лог.

1. Аналитическая часть

В данной лабораторной работе реализована некая функцию хеширования строки, которая состоит из трех последовательных действий: применения первой хеш-функции, применения второй хеш-функции и применения третьей хеш-функции. Если необходимо вычислить хеш для какого-то массива строк, то можно использовать конвейерную обработку данных. Таким образом задача будет решена эффективнее, чем при последовательном применении алгоритмов к массиву значений.

Конвейер будет состоять из четырех уровней. Обработанные данные передаются последовательно с одного уровня (одной ленты) конвейера на следующий (следующую ленту). Далее на каждом уровне осуществляется обработка данных, занимающая определенное время. Для каждой ленты создается своя очередь задач, в которой хранятся все необработанные строки. На последнем уровне конвейера обработанные объекты попадают в пул обработанных задач.

Уровни конвейера:

- 0 уровень — генерация входных данных в первую очередь;
- 1 уровень(лента) — применение первого хеша к строкам из первой очереди , задержка + задержка 1 с, запись результата во 2 очередь;
- 2 уровень(лента) — применение второго хеша к строкам 2 очереди + задержка 3 с, запись результата в 3 очередь;
- 3 уровень(лента) — применение третьего хеша к строкам 3 очереди + задержка 1,5 с, запись результата в пул обработанных задач.

Поскольку запись в очередь и извлечение из очереди это не атомарные операции, необходимо создать их таковыми путем использования мьютексов (по одному на одну очередь) и критических секций, чтобы избежать ошибок в ситуации гонок.

2. Конструкторская часть

2.1 Описание архитектуры ПО

В конвейере 3 основные ленты, содержание их работы описано выше, в аналитической части отчета. Каждой ленте выделен свой поток, в котором она выполняется. В главном потоке (функция main) создаются три рабочих потока: по одному на каждую ленту.

Для каждой ленты есть своя очередь, однако с ней могут работать все потоки, поэтому при доступе к элементам очереди необходимо блокировать доступ для других потоков. Для реализации доступа из разных потоков используются мьютексы, по одному для каждой очереди и для результирующего массива создан.

Также в главном потоке генерируется массив входных данных и заполняется первая очередь (уровень 0). Для моделирования ситуации постепенного поступления данных, первая очередь заполняется с задержкой.

В рабочих процессах считывается по одному элементу из соответствующей очереди, выполняется вызов обрабатывающих функций, замеры времени и задержка по времени. После обработки текущей задачи, рабочий процесс записывает результат в следующую очередь или в результирующий массив. Также выполняется запись в лог-файл.

Схема работы конвейера представлена на рис. 1.

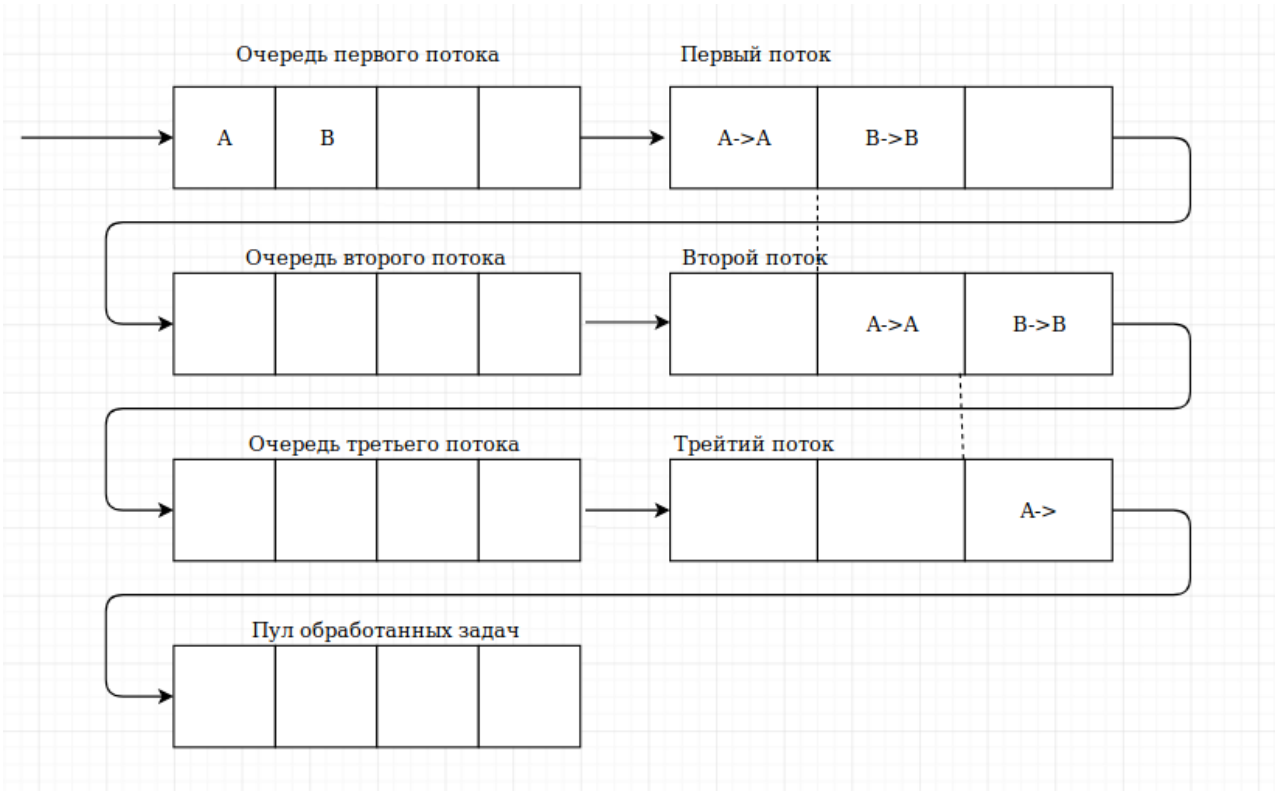

Рис 1. Схема работы конвейерной обработки

2.2 Разработка алгоритмов

На рисунках 2-3 представлены схемы алгоритмов работы конвейерной обработки.

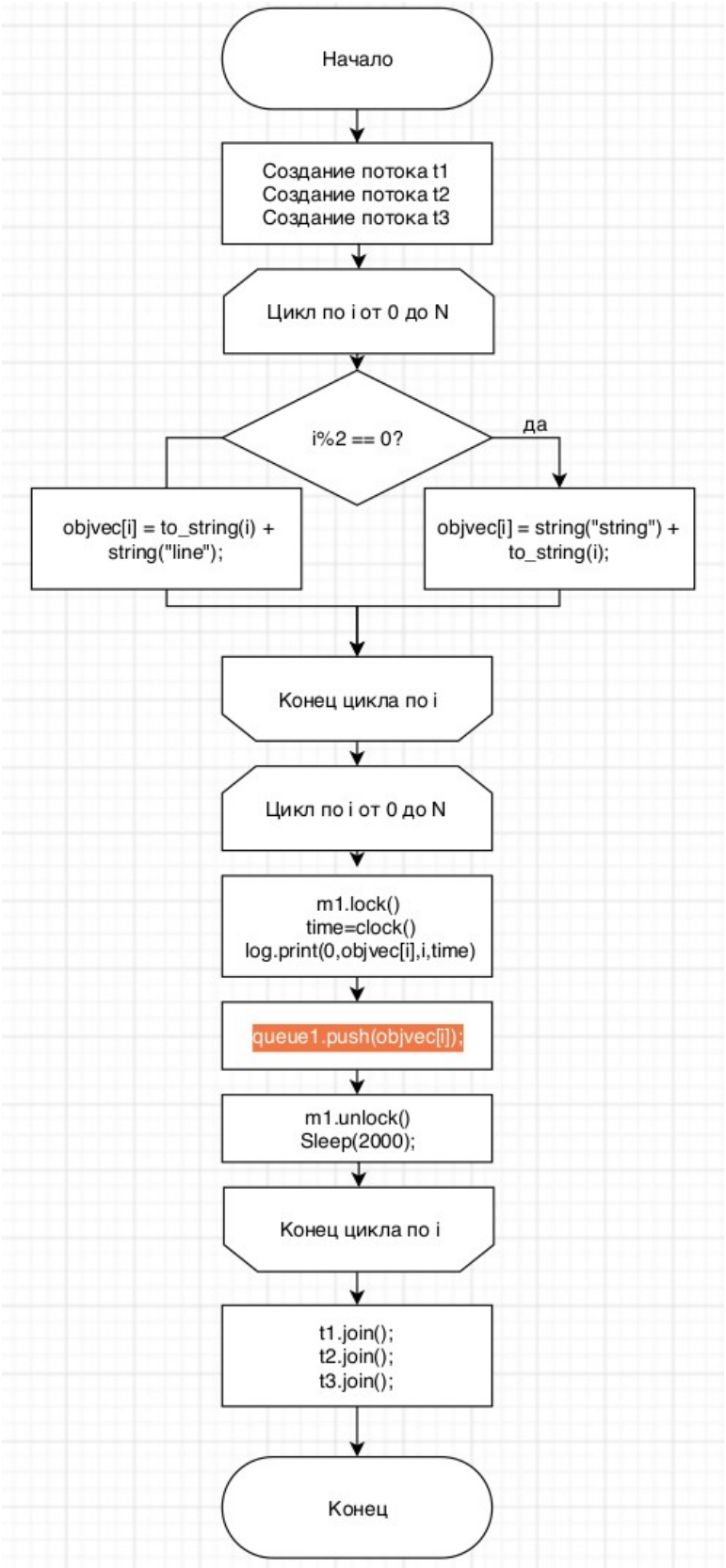

Рис 2. Схема главного потока

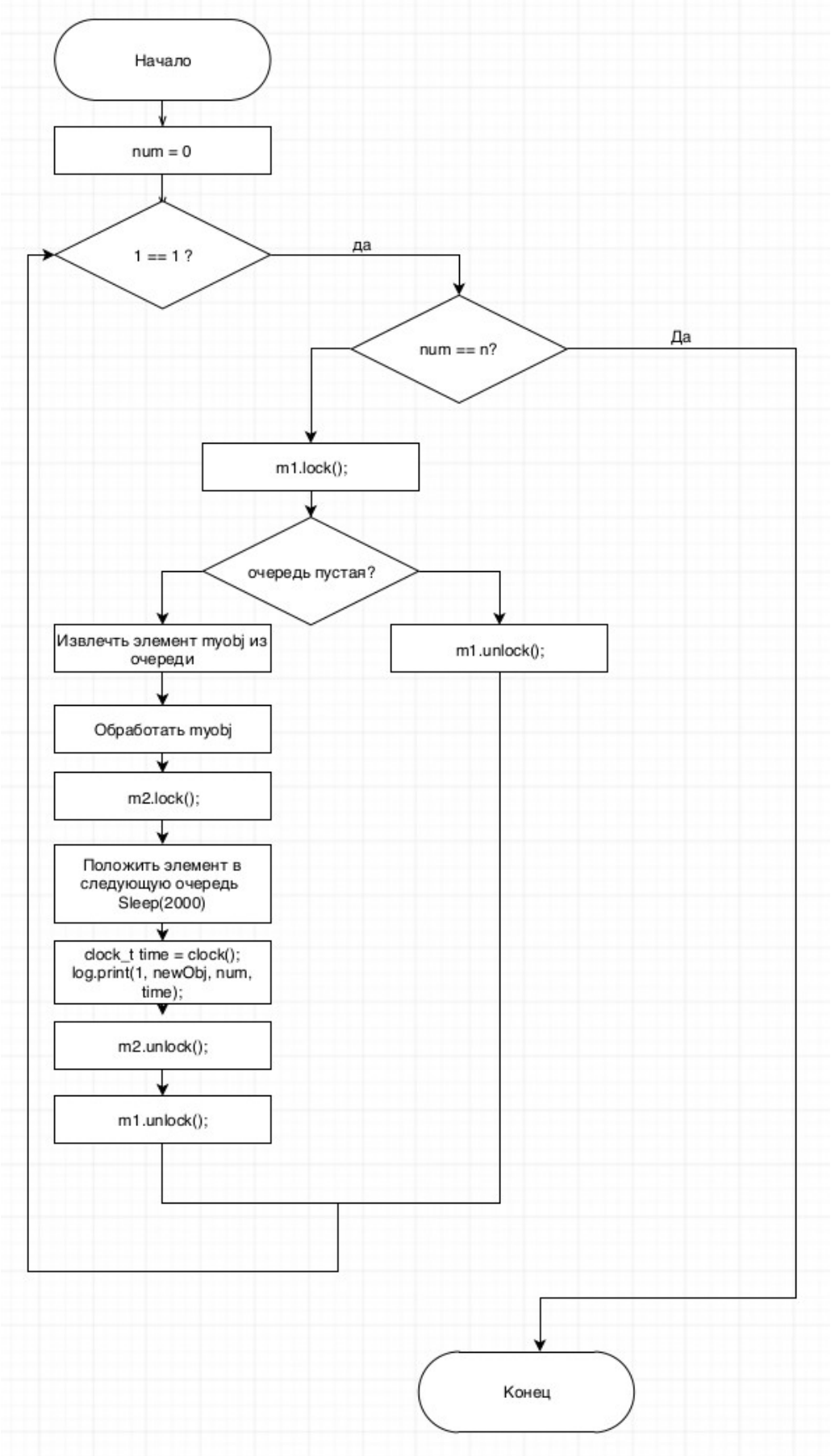

Рис 3. Схема работы рабочего процесса

3 Технологическая часть

3.1 Средства реализации

В качестве языка программирования был выбран C++ (компилятор g++) в связи с его широким функционалом и быстротой работы. Среда разработки - Qt. Для работы с потоками использовалась библиотека thread. Также использована библиотека mutex для контроля обращения разных потоков к одному участку данных. Для реализации очереди выбрана библиотека queue. Для замеров времени использована библиотека ctime.

3.2 Листинг кода

На листингах 1-4 представлены фрагменты кода полученной программы.

Листинг 1. Реализация 1 уровня конвейера

```
void SingleHash() {
    int num = 0;
    while (1) {
        if (num == n)
            break;
        m1.lock();
        if (queue1.empty()) {
            m1.unlock();
            continue;
        }
        input_t myObj = queue1.front();
        queue1.pop();
        input_t newObj = myHash1(myObj);
        m2.lock();
        queue2.push(newObj);
        Sleep(1000);
        clock_t time = clock();
        log.print(1, newObj, num, time);
        m2.unlock();
        m1.unlock();
        num++;
    }
}
```

Листинг 2. Реализация 2 уровня конвейера

```
void MultiHash() {
    int num = 0;
    while (1) {
        if (num == n)
            break;
        m2.lock();
        if (queue2.empty()) {
            m2.unlock();
            continue;
        }
        input_t myObj = queue2.front();
        queue2.pop();
```

```
        input_t newObj = myHash2(myObj);
        m3.lock();
        queue3.push(newObj);
        Sleep(3000);
        clock_t time = clock();
        log.print(2, newObj, num, time);
        m3.unlock();
        m2.unlock();
        num++;
    }
}
```

Листинг 3. Реализация 3 уровня конвейера

```
void Result() {
    int num = 0;
    while (1) {
        if (num == n)
            break;
        m3.lock();
        if (queue3.empty()) {
            m3.unlock();
            continue;
        }
        input_t myObj = queue3.front();
        queue3.pop();
        input_t newObj = myHash3(myObj);
        resm.lock();
        res.push_back(newObj);
        Sleep(1500);
        clock_t time = clock();
        log.print(3, newObj, num, time);
        resm.unlock();
        m3.unlock();
        num++;
    }
}
```

Листинг 4. Основная функция программы

```
int main()
{
    cout << "Hello World!" << endl;

    f = fopen("res.txt", "w");
    n = 5;
    objvec.resize(n);

    thread t1(SingleHash);
    thread t2(MultiHash);
    thread t3(Result);
    main_time = clock();
    for (int i = 0; i < n; i++) {
        if (i % 2 == 0) {
            objvec[i] = string("string") + to_string(i);
        }
```

```
        else { objvec[i] = to_string(i) + string("line"); }
    }

    for (int i = 0; i < n; i++) {
        m1.lock();
        clock_t time = clock();
        log.print(0, objvec[i], i, time);
        queue1.push(objvec[i]);
        m1.unlock();
        Sleep(2000);
    }

    t1.join();
    t2.join();
    t3.join();
    fclose(f);
    return 0;
}
```

4 Экспериментальная часть

4.1 Анализ результатов

На выходе программа возвращает лог-файл, в котором находится информация о процессе выполнения программы. Содержимое лог-файла представлено в листинге 5.

Листинг 5. Содержимое лог-файла

```
[0]  item0 time: 9 (0)  value: string0
[1]  item0 time: 1019 (1010)  value: DFE=C=W~DGG@GB]
[0]  item1 time: 2018 (999)  value: 1line
[2]  item0 time: 4027 (2009)  value:
jki`e^'Mbdc[a[%_ehhahc.Vm!"l$p<~eijdlh4]%*,'0J_[^^W^Y$Lcghbjf2
[1]  item1 time: 5037 (1010)  value: R><B:~R?>E>
[0]  item2 time: 5037 (0)  value: string2
[3]  item0 time: 5537 (500)  value: resultJKI@E>W-BDC;A;U?
EHHAHC^6MQRLTPl^EIJDLHd=UZ\W`]*?;>>7>9T,CGHBJFbhash
[2]  item1 time: 8545 (3008)  value: (c`e\O"^\bZ_#`_f_T)ggoi~#aaicY/no(#_iVU\UJo]]e_
[1]  item2 time: 9555 (1010)  value: DFE=C=Y~DGG@GB_
[0]  item3 time: 9555 (0)  value: 3line
[3]  item1 time: 10054 (499)  value: resultXC@E</R><B:?S@?F?4YGGOI^SAAIC9_NOXS?
I65<5*O==E?hash
[2]  item2 time: 13063 (3009)  value: jki`e^)Mbdc[a['_ehhahc0Vm!"l$p>~eijdlh6]%*,'0-
L_[^^W^Y&Lcghbjf4
[1]  item3 time: 14072 (1009)  value: T><B:~T?>E>
[0]  item4 time: 14072 (0)  value: string4
[3]  item2 time: 14572 (500)  value: resultJKI@E>Y-BDC;A;W?
EHHAHC`6MQRLTPn^EIJDLHf=UZ\W`],?;>>7>9V,CGHBJFdhash
[2]  item3 time: 17580 (3008)  value: *c`e\O$^\bZ_%`_f_T+ggoi~%aaicY1no(#_kVU\UJ!]]e_
```

```
[1]  item4 time: 18590 (1010)  value: DFE=C=[~DGG@GBa
[3]  item3 time: 19090 (500)  value: resultZC@E</T><B:?U@?F?4[GGOI^UAAIC9aNOXS?
K65<5*Q==E?hash
[2]  item4 time: 22108 (3018)  value: jki`e^+Mbdc[a()_ehhahc2Vm!"l$p@~eijdlh8]%*,'0-
N_[^^W^Y(Lcghbjf6
[3]  item4 time: 23617 (1509)  value: resultJKI@E>[-BDC;A;Y?
EHHAHCb6MQRLTPp^EIJDLHh=UZ\W`].?;>>7>9X,CGHBJFfhash
```

Для наглядности в листинге цветами выделенны логи о разных элементах.

Лог файл состоит из записей отсортированных по времени с начала запуска приложения. В первом столбце указан номер лининии обработки, затем номер текущего элемента, затем время с начала работы программы и время выполнения данного этапа в миллисекундах, а также значение хеша на текущем этапе.

По листингу можно проследить выполнение каждого этапа и изменение значений хеша при обработке на каждой из лент. Также можно проследить, что обработка выполняется параллельно. Например, 2 линия не ждет полного завершения работы первой прежде, чем начать работу, а начинает выполнение как только в очередь поступает первый элемент.
Так как обработка на каждой линии занимает разное количество времени, в листинге можно увидеть, насколько она эфективнее. Если бы мы обрабатывали каждый элемент последовательно, то потребовалось бы $(1000+3000+1500)*5=27500$ миллисекунд. Однако алгоритм выполнился за 23617 миллисекунд, что дает выигрыш в 4 секунды.

4.2 Вывод

Алгоритм конвейерной обработки эффективен для ситуаций, когда каждая линия конвейера требует разное время на обработку задачи. Однако в случае, когда все линии обрабатывают задачи за одинаковое время, алгоритм не будет давать существенного выигрыша.

Заключение

В ходе лабораторной работы был изучен и реализован вычислительный конвейер с использованием методов распараллеливания процессов. Были выявлены оптимальные для конвейерной обработки параметры входных задач: сбалансированное время обработки данных на всех лентах (уровнях) конвейера.