Государственное образовательное учреждение высшего профессионального образования
«Московский государственный технический университет имени Н. Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Дисциплина: анализ алгоритмов

Лабораторная работа №5

Студент: Власова Екатерина, ИУ7-54

2019 г.

Содержание

Введение

При выполнении программой одной операции за единицу времени, можно сказать, что тысячу операций устройство выполнит за тысячу единиц. Для сокращения времени работы программы используются параллельные вычисления, одновременно выполняющие некоторые подфункции. Конвейерная обработка состоит в выделении отдельных этапов выполнения общей операции. Каждый этап, выполнив свою работу, передаёт результат следующему, одновременно принимая новую порцию данных. Совмещение прежде разрозненных во времени операций определенно положительно влияет на скорость обработки. Например, в операции можно выделить пять микроопераций, каждая из которых выполняется за одну единицу времени.

1 Аналитическая часть

Описание рассматриваемого алгоритма.

1.1 **Описание алгоритма**

Для реализации конвейерной обработки с помощью потоков было выделено три этапа, на каждом из которых производится своя операция обработки данных. Каждый из трёх потоков выполняет свою операцию и затем передаёт результат следующему потоку (за исключением третьего потока, он является завершающим и ничего дальше не передаёт).

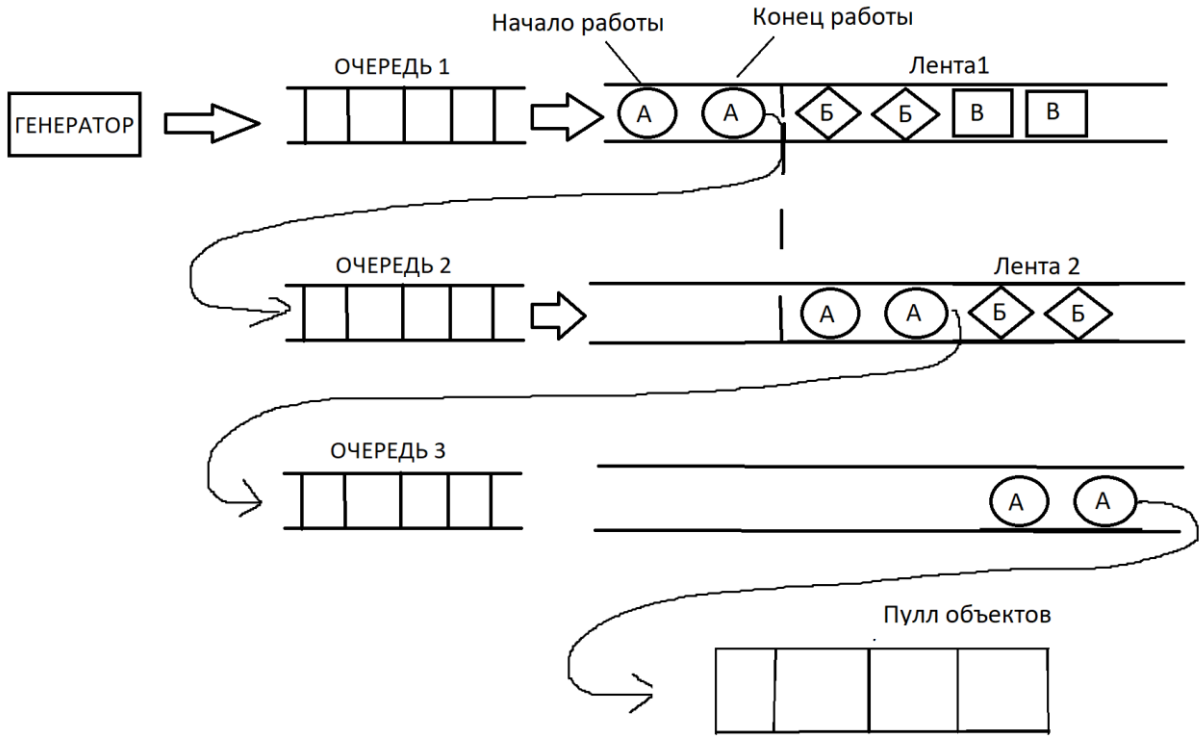

Рисунок 1. Схема работы конвейера

1.2 Задание на выполнение лабораторной работы

- Получить навык организации асинхронного взаимодействия потоков на примере конвейерной обработки данных;
- Обработать и описать технический процесс стадийной обработки данных.

1.3 Вывод по аналитической части

Поставлена задача на выполнение лабораторной работы, описана схема работы конвейера.

2 Конструкторская часть

В данном разделе рассмотрен способ реализации конвейера.

2.1 Способ организации конвейера

Конвейер состоит из трёх потоков для трёх последовательных задач и трёх очередей, с помощью которых потоки взаимодействуют друг с другом и передают данные. Ленты ограничены по количеству элементов, и сгенерированные таски, не поместившиеся в ленту, «теряются». Рабочие потоки забирают задачу из своей очереди, ставят отметку времени начала обработки, проводят обработку, ставят отметку об окончании обработки и передают задачу в очередь следующей ленты. Первый поток генерирует числа в подаваемом ему массиве, второй поток сортирует числа в этом массиве, а третий выводит массив в файл «fileN.txt», где N – номер обрабатываемой задачи. В данной работе используется сортировка вставками.

2.2 Сортировка вставками

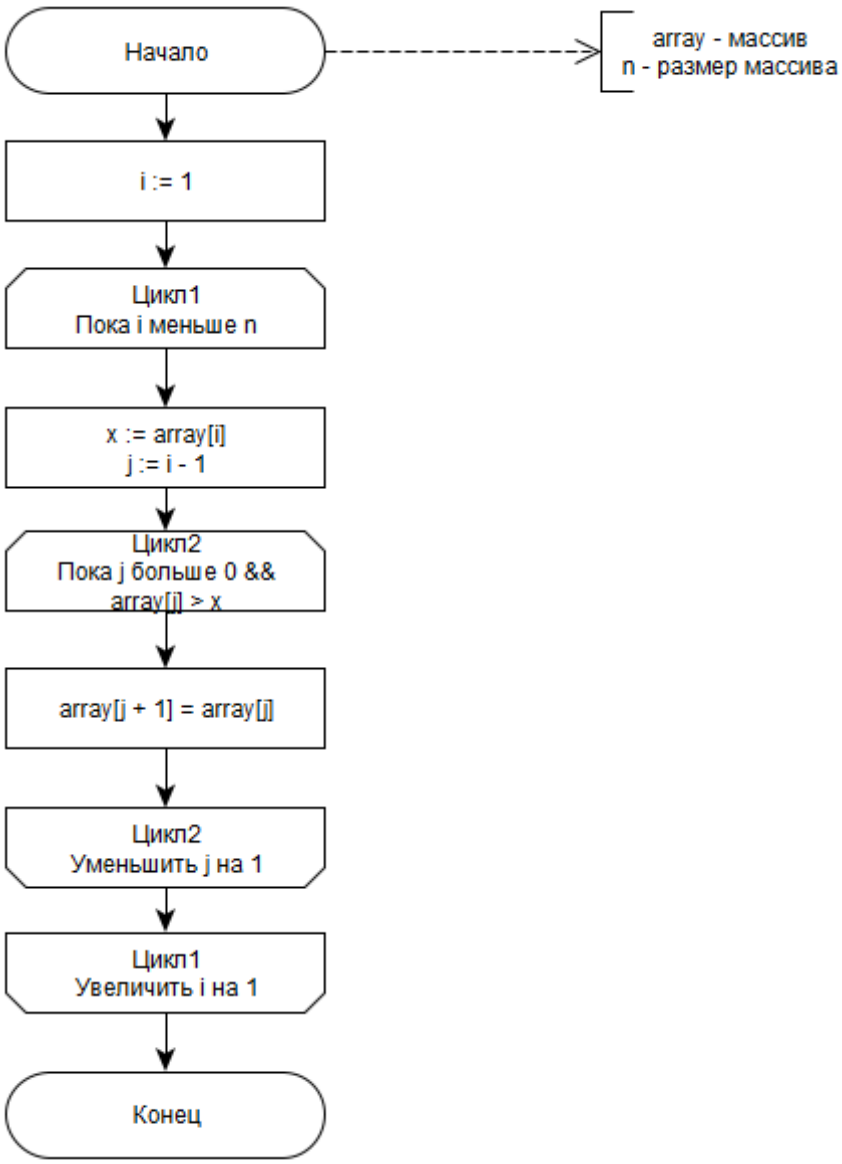

Рисунок 2.2 Сортировка вставками

2.3 Первая обрабатывающая лента

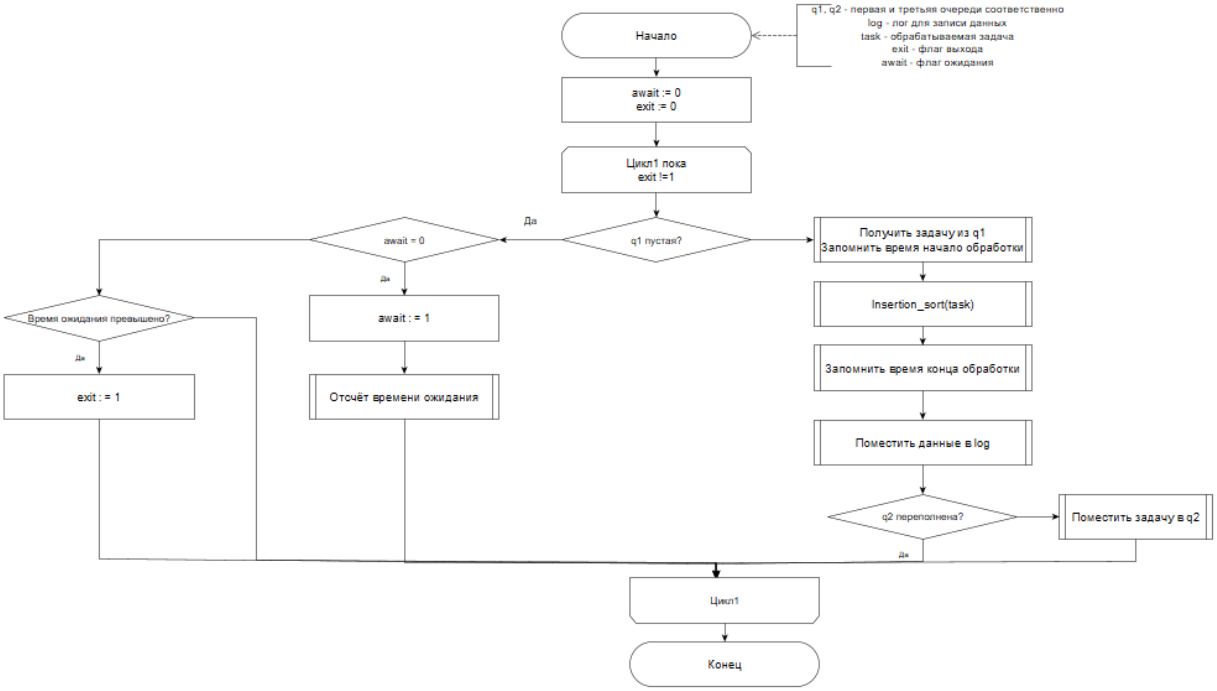

Рисунок 2.3 Первая обрабатывающая лента

2.4 Вторая обрабатывающая лента

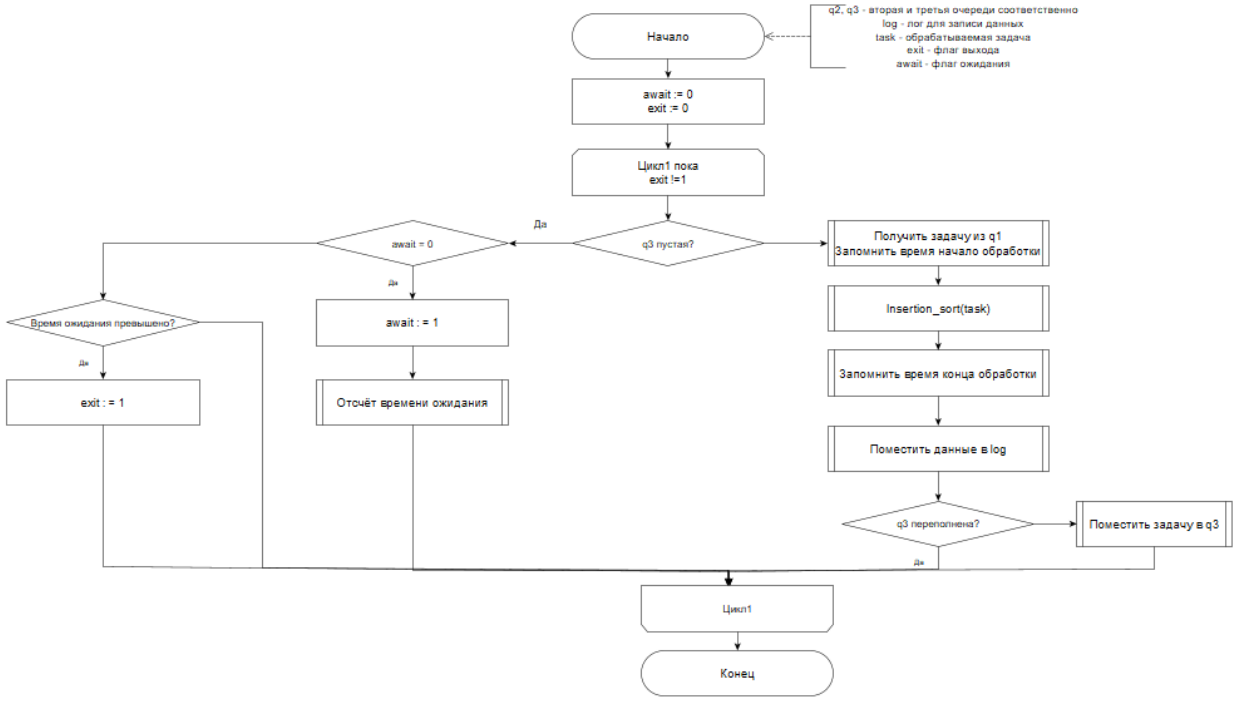

Рисунок 2.4 Вторая обрабатывающая лента

2.5 Третья обрабатывающая лента

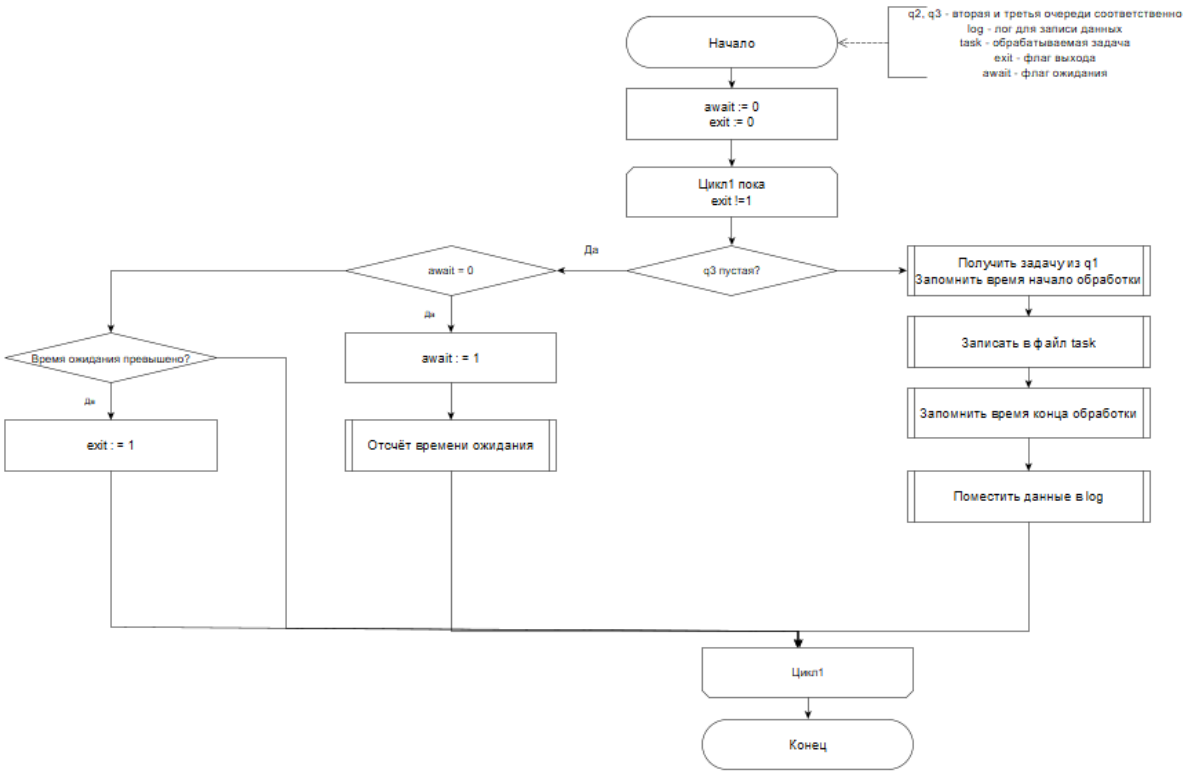

Рисунок 2.5 Третья обрабатывающая лента

2.2 Выводы по конструкторскому разделу

Был рассмотрен способ реализации конвейера: строение лент и рассматриваемый алгоритм сортировки.

3 Технологическая часть

В данном разделе был сделан выбор используемого языка программирования, предоставлен код алгоритмов.

3.1 **Требования к программному обеспечению**

В программе задаётся количество задач на выполнение, после чего генератор в режиме реального времени генерирует их и кладёт в первую очередь. Очереди ограничены по количеству элементов: «не вмещающиеся» задачи пропадают. После отработки всех задач выводится лог объектов.

3.2 **Средства реализации**

В качестве языка программирования в данной работе был выбран C++, его стандарт представлен в [7]. Для работы с потоками используется библиотека thread [11], а для замеров времени – chrono [10].

3.3 Листинг кода

3.3.1 Первая обрабатывающая лента

Листинг 0

```
void executer(myqueue& q1, myqueue& q2, Log &log)
{
    int tape_id = ++tape_number;
    long long all_waiting_time = 0;
    int number_of_waitings = 0;
    std::chrono::time_point<std::chrono::steady_clock> start_awaiting_time;
    bool awaiting = false;
    bool exit = false;

    std::mt19937 gen1(time(0));
    std::uniform_int_distribution<int> uid(500, 1000);
    std::uniform_int_distribution<int> uid2(0, 1000);

    while(!exit){

        auto task = q1.getTask();

        if (task.number != -1)
        {
            auto time_st = time_point(start);

            int len = uid(gen1);
            task.lenght = len;
            task.data = new int[len];
            for (int i = 0; i < len; i++)
                task.data[i] = uid2(gen1);
            auto time_end = time_point(start);

            data_t data = init(tape_id, task.number, task.lenght, time_st,
time_end);
            log.set(data);


            q2.set(task);
            if (awaiting)
            {
                awaiting = false;
                all_waiting_time += time_point(start_awaiting_time);
                number_of_waitings += 1;
            }
        }
        else
        {
            if (!awaiting)
            {
                awaiting = true;
                start_awaiting_time = std::chrono::steady_clock::now();
            }
            else  if (number_of_waitings > 0 &&
time_point(start_awaiting_time) > 2 * (all_waiting_time /
number_of_waitings))
            {
                exit = true;
            }
        }
    }
    return;
}
```

3.3.2 Вторая обрабатывающая лента

Листинг 1

```
void executer2(myqueue& q1, myqueue& q2, Log &log)
{
    int tape_id = ++tape_number;

    std::chrono::time_point<std::chrono::steady_clock> start_awaiting_time;
    bool awaiting = false;
    bool exit = false;
    long long all_waiting_time = 0;
    int number_of_waitings = 0;
    std::mt19937 gen1(time(nullptr));
    std::uniform_int_distribution<> urd(1750, 2250);

    while(!exit){

        auto task = q1.getTask();

        if (task.number != -1)
        {
            auto time_st = time_point(start);

            insertion_sort(task);

            auto time_end = time_point(start);

            data_t data = init(tape_id, task.number, task.lenght, time_st,
time_end);
            log.set(data);


            q2.set(task);

            if (awaiting)
            {
                awaiting = false;
                all_waiting_time += time_point(start_awaiting_time);
                number_of_waitings += 1;
            }
        }
        else
        {
            if (!awaiting)
            {
                awaiting = true;
                start_awaiting_time = std::chrono::steady_clock::now();
            }
            else  if (number_of_waitings > 0 &&
time_point(start_awaiting_time) > 2 * (all_waiting_time /
number_of_waitings))
            {
                exit = true;
            }

        }
    }
    return;
}
```

3.3.3 Третья обрабатывающая лента

Листинг 2

```
void executer3(myqueue& q1, Log &log)
{
    int tape_id = ++tape_number;

    std::chrono::time_point<std::chrono::steady_clock> start_awaiting_time;
    bool awaiting = false;
    long long all_waiting_time = 0;
    int number_of_waitings = 0;
    bool exit = false;

    std::mt19937 gen1(time(nullptr));
std::uniform_int_distribution<> urd(1750, 2250);

    while(!exit){
        auto task = q1.getTask();

        if (task.number != -1)
        {
            auto time_st = time_point(start);
            std::string strin = "file" + std::to_string(task.number) +
".txt";
            std::ofstream file (strin);
            for(int i = 0; i < task.lenght; i++)
            {
                file << task.data[i];
                file << " ";
            }
            auto time_end = time_point(start);


            data_t data = init(tape_id, task.number, task.lenght, time_st,
time_end);
            log.set(data);

            if (awaiting)
            {
                awaiting = false;
                all_waiting_time += time_point(start_awaiting_time);
                number_of_waitings += 1;
            }
        }
        else
        {
            if (!awaiting)
            {
                awaiting = true;
                start_awaiting_time = std::chrono::steady_clock::now();
            }
            else  if (number_of_waitings > 0 &&
time_point(start_awaiting_time) > 2 * (all_waiting_time /
number_of_waitings))
            {
                exit = true;
            }
            }
        }
 }
```

3.3.4 Генератор

Листнинг 3

```
void generator(myqueue &q)
{
    int counter = 0;

    auto t = static_cast<int>(time(nullptr));
    std::mt19937 gen1(t);
    std::uniform_int_distribution<> urd(1, TASK_NUMBER);
    std::uniform_int_distribution<> urd1(100, 500);

    while (counter < TASK_NUMBER)
    {
        auto tmp_tasks_number = urd(gen1);

        for (int i=0; i < tmp_tasks_number and counter < TASK_NUMBER; i++)
        {
            Task task = init(++counter);

            q.set(task);
        }

        auto sleep_time = urd1(gen1);
        std::this_thread::sleep_for(std::chrono::milliseconds(sleep_time));
    }
    return;
}
```

3.4 Выводы по технологическому разделу

Был выбран язык программирования, описаны требования к ПО. Представлен код параллельного конвейера.

4 Экспериментальная часть

В данном разделе представлены примеры работы программы.

4.1 Примеры работы

Представлены примеры работы программы, где:

1. Tape – номер ленты;
2. Task – номер задачи;
3. Lenght – длина массива;
4. start и end – начало и конец обработки задачи лентой соответственно.

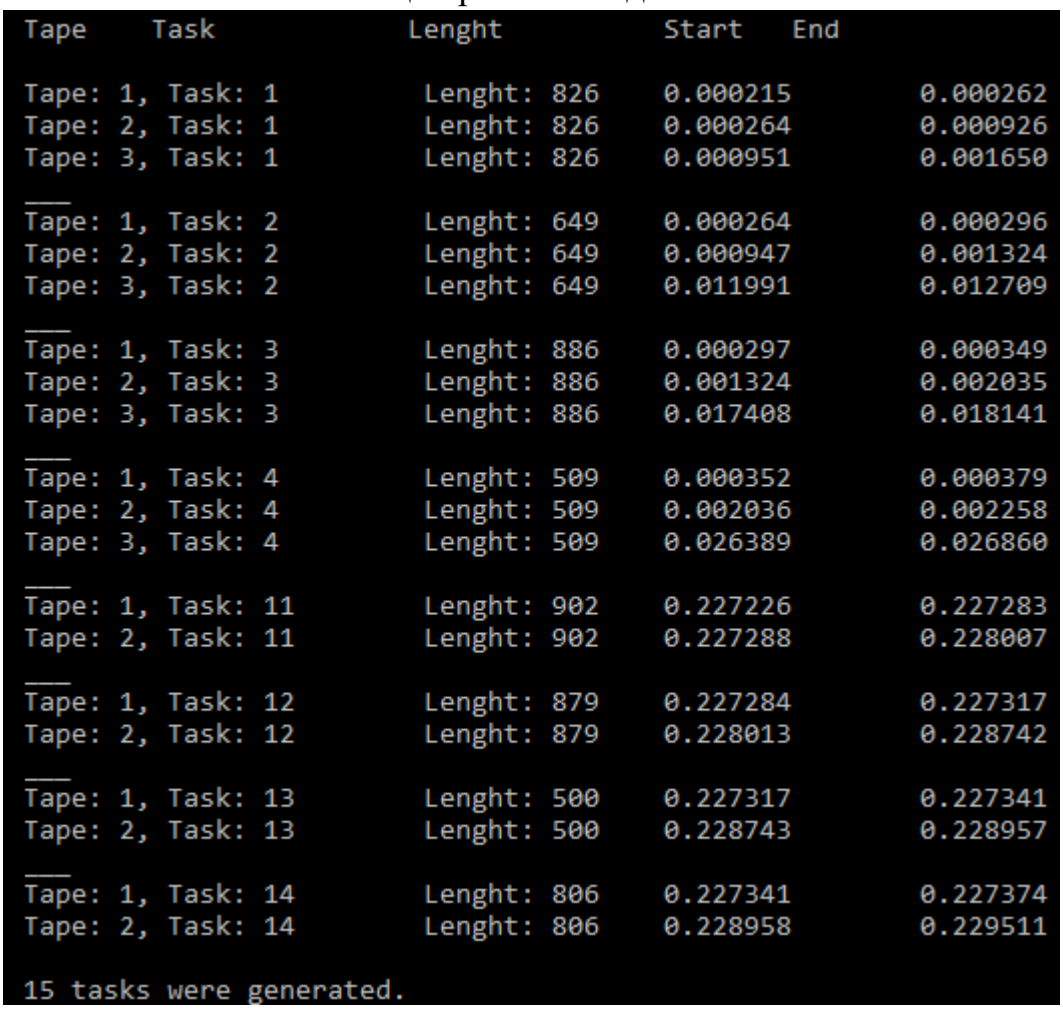

Рисунок 4.1.1 Пример работы программы

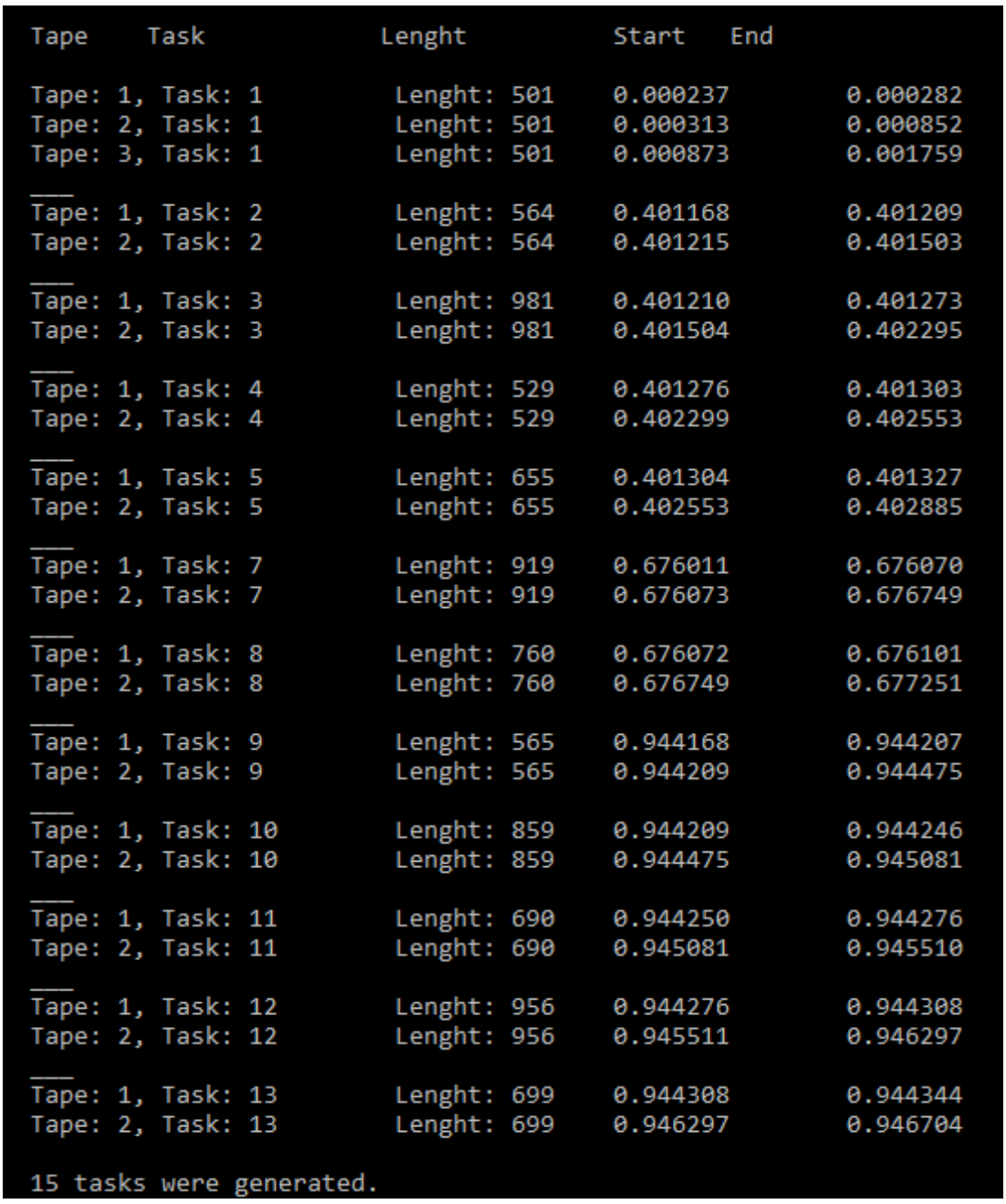

Рисунок 4.1.2 Пример работы программы

4.4 Выводы по экспериментальной части

Исходя из представленных примеров работ программы: если время обработки на первой ленте больше времени генерации заданий и очередь ограничена по количеству принимаемых задач, то некоторые объекты могут «потеряться» и так и не выполниться. Также не гарантируется полная обработка задачи всеми лентами: если предыдущая лента обрабатывает задачу в два раза дольше, чем время ожидания следующей очереди, то следующая лента завершает свою работу. Например, на рисунке 4.1.2 представлена такая ситуация: на выполнении второй задачи завершила работу первая лента. Это связано с тем, что размерность массива, первого поступившего на ленту, меньше размера второго массива. И время обработки задачи первой и второй лентой оказалось в два раза больше, чем время ожидания третьей лентой.

Заключение

Описан технический процесс параллельной конвейерной обработки данных. В ходе выполнения данной лабораторной работы был реализован параллельный конвейер, выполняющий задачу последовательно в нескольких потоках.

Исходя из представленных примеров работ программы: если время обработки на первой ленте больше времени генерации заданий и очередь ограничена по количеству принимаемых задач, то некоторые объекты могут «потеряться» и так и не выполниться. Также не гарантируется полная обработка задачи всеми лентами: если предыдущая лента обрабатывает задачу в два раза дольше, чем время ожидания следующей очереди, то следующая лента завершает свою работу. Например, на рисунке 4.1.2 представлена такая ситуация: на выполнении второй задачи завершила работу первая лента. Это связано с тем, что размерность массива, первого поступившего на ленту, меньше размера второго массива. И время обработки задачи первой и второй лентой оказалось в два раза больше, чем время ожидания третьей лентой.

Список использованной литературы

1. Pete Becker, Working Draft, Standard for Programming Language C++, 2011.
2. Алексей Куканов, Многопоточность и параллелизм в C++, 2015.
3. Лацис А.О. Параллельная обработка данных, 2010.
4. Р. Миллер, Л. Боксер Последовательные и параллельные алгоритмы: общий подход, 2006.
5. Воеводин В. В., Воеводин А. В., Параллельные вычисления, 2002.
6. Э. Уильямс. Параллельное программирование на C++ в действии. Практика разработки многопоточных программ, 2012.
7. Стандарт C++, [Электронный ресурс] URL: https://isocpp.org/std/the-standard
8. Multi-threading, [Электронный ресурс] URL: http://www.cplusplus.com/reference/multithreading/
9. Введение в технологии параллельного программирования [Электронный ресурс] URL: https://software.intel.com/ru-ru/articles/writing-parallel-programs-a-multi-language-tutorial-introduction/
10. Библиотека chrono [Электронный ресурс] URL: https://docs.microsoft.com/ru-ru/cpp/standard-library/chrono?view=vs-2017
11. Библиотека std::thread, [Электронный ресурс] URL: https://en.cppreference.com/w/cpp/thread/thread
12. Библиотека mutex [Электронный ресурс] URL: https://docs.microsoft.com/ru-ru/cpp/standard-library/mutex-class-stl?view=vs-2019