Государственное образовательное учреждение высшего профессионального образования
“Московский государственный технический университет имени Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа № 6

ТЕМА
«Конвейер»

Студент группы ИУ7-54,
Лозовский Алексей

2019 г.

Содержание

Введение

В данной работе речь пойдет о работе с разновидностью параллельного программирования - асинхронным программированием. При обработке данных могут возникать ситуации, когда необходимо обработать множество данных последовательно несколькими алгоритмами. В этом случае удобно использовать конвейерную обработку данных. Это может быть полезно при следующих задачах:

- шифровании данных;
- сортировки и фильтрации данных;
- и др.

1 Аналитическая часть

Одним из способов обработки данных является конвейерная обработка. С ее помощью можно реализовать разнесение технических операций на разные ленты конвейера.

1.1 Описание алгоритмов

Генератор в режиме реального времени создаются объекты(задачи) и передает их на 1 ленту. **Объект** - данные, которые должны быть обработаны. Рабочие потоки забирают объект из своей очереди, ставят отметку о начале работы, проводят обработку, фиксируют время окончания и передают в очередь следующей ленты, либо в случае последней ленты в пулл обработанных задач. По итогам работы с задачами обрабатывается пул объектов и формируется лог.

Так как все ленты работают параллельно, несколько задач могут выполняться одновременно, при этом каждый объект выполняется последовательно на каждой очереди. На рис.1 отображена схема работы конвейера.

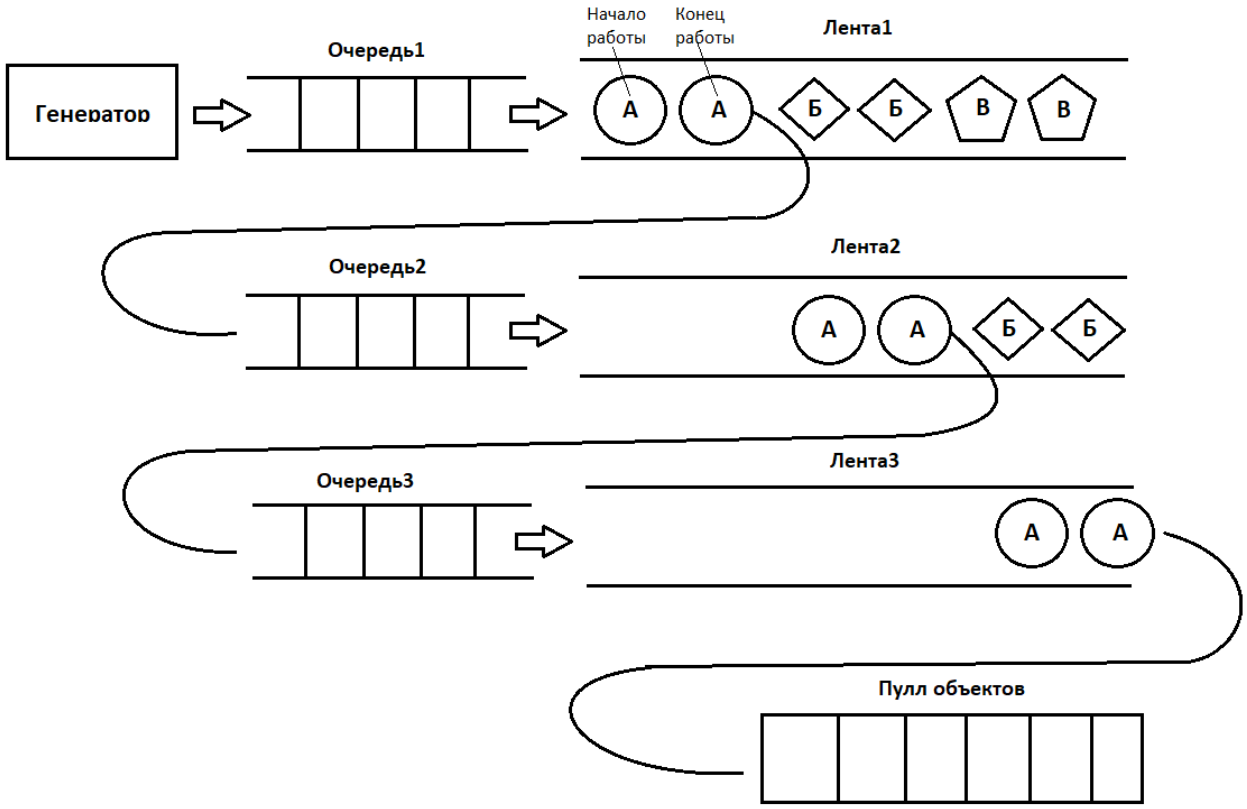

Рис. 1: Схема работы конвейера.

1.2 Задание на выполнение лабораторной работы

Цель: Получить навык организации асинхронного взаимодействия потоков на примере конвейерной обработки данных.

Задачи:

- изучить работу конвейера;
- описать технический процесс стадийной обработки данных;
- спроектировать структуру ПО;
- описать механизм обмена данными;
- реализовать конвейерную обработку задач.

1.3 Вывод по аналитическому разделу

По итогам аналитического раздела было выполнено следующее:

- изучена и описана работа конвейера;
- поставлены цель и задачи для выполнения лабораторной работы.

2 Конструкторская часть

В конструкторской части необходимо сделать следующее:

- определить структуру кода и способ представления данных;
- разработать схему механизма обмена данных;

2.1 Структура кода и представление данных

Для написания программы был выбран структурный подход. Для заданий и лога должны быть выделены отдельные классы для удобства использования и наглядности в коде.

2.2 Разработка алгоритмов

2.2.1 Обработка данных в ленте

На рис. 2 отображена работа ленты с данными.

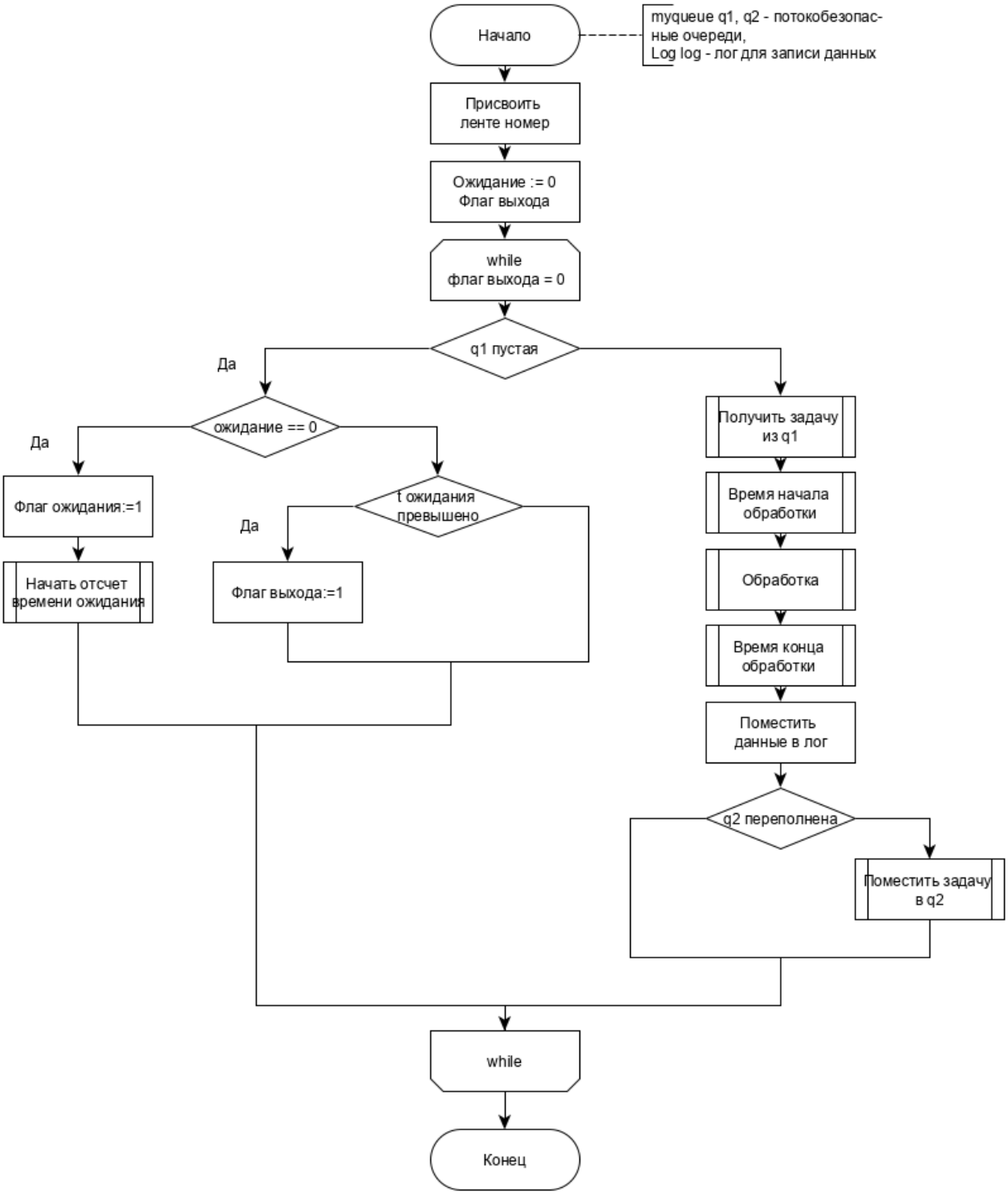

Рис. 2: Схема работы ленты.

2.2.2 Описание работы алгоритма

Каждая лента берет из очереди задачу на обработку, после чего считает время начала и конца работы и отправляет полученные данные в лог, а задачу в следующую очередь. В случае пустой очереди начинается отсчет времени ожидания – если время ожидания превышает удвоенное среднее ожидание - лента завершает свою работу. При попытке подать задачу в переполненную очередь - задача теряется.

2.3 Выводы по конструкторскому разделу

Таким образом, была определена структура кода и способ представления данных, также была разработана схема обмена данными.

3 Технологическая часть

3.1 Требования к программному обеспечению

Программа должна выводить в консоль сгенерированный в ходе работы лог, в котором должны храниться номер ленты, номер задания, время начала работы задания на ленте, время окончания.
Доступ к каждой очереди должен быть организован в монопольном режиме, чтобы избежать ошибок в ситуации "гонок".
Каждый поток должен обработать все полученные задачи.

3.2 Средства реализации

Поскольку программа реализована в структурном стиле, из языков, поддерживающих его был выбран C++, стандарта 2011 года[2], так как на нем имеется наибольший опыт работы. Для реализации потоков была выбрана библиотека <thread> с целью ознакомления с ее возможностями. Для замера времени будем использовать библиотеку <chrono>, так как библиотека <ctime> неверно считает время при параллельных вычислениях. Для реализации монопольного режима будем использовать библиотеку <mutex>.

3.3 Листинг кода

Ниже представлены листинги кода для класса очереди(Листинг 1, 2) и для функции обмена и обработки данных(Листинг 3).

Листинг 1: Класс очереди для работы с потоками

```
class myqueue
{
    public:
        myqueue();

        void set(Task task);
        Task getTask();
        bool isEmpty();
        bool isFull();

    private:
        std::queue<Task> queue;
        std::mutex mutex;
};
```

Листинг 2: Методы класса myqueue

```
myqueue::myqueue(){}


void myqueue::set(Task task)
{
    std::lock_guard<std::mutex> lock(mutex);
```

```
    if (!isFull())
        queue.push(task);
}


bool myqueue::isFull()
{
    return queue.size() > QUEUE_MAX_SIZE ? 1 : 0;
}


bool myqueue::isEmpty()
{
    return queue.empty();
}


Task myqueue::getTask()
{
    std::lock_guard<std::mutex> lock(mutex);

    Task task;
    task.number = -1;

    if (!isEmpty()){
        task = queue.front();
        queue.pop();
}

return task;
}
```

Листинг 3: Методы класса myqueue

```
void executer(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;

    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);

            auto working_time = urd(gen1);
            std::this_thread::sleep_for(std::chrono::milliseconds(working_time));

            auto time_end = time_point(start);

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

            q2.set(task);

            awaiting = false; // сбросфлагаожидания
        }
        else
        {
            if (!awaiting){
                awaiting = true;
                start_awaiting_time = std::chrono::steady_clock::now();
                }
        }
        else
        {
            auto awaiting_time = time_point(start_awaiting_time);

            if (time_is_out(awaiting_time))
                exit = true;
        }
    }
    return;
}
```

3.4 Выводы по конструкторскому разделу

Были описаны требования к ПО, приведен листинг кода с реализацией фукнции для подачи в поток, а также был выбран язык программирования и бибилотека для работы с потоками и замером времени работы.

4 Экспериментальная часть

4.1 Постановка эксперимента

Установим, что генератор может создать не более 15 заданий. Ограничим очередь по количеству содержащихся объектов до 4. Пусть генератор будет подавать данные в очередь с некоторой периодичностью, при этом за короткий промежуток может быть сгенерировано как 2 объекта так и 15. Также установим время обработки на ленте больше чем время генерации задач.

4.2 Примеры работы

Ниже в таблице 1 представлен результат работы программы при условиях, описанных выше при обработке 15 задач. Tape N - номер ленты, на которой была обработана задача Task M, где M - номер задачи, а также время начала работы в ленте - Start Time и конца - End Time.

Таблица 1: Лог работы программы.

Tape N	Task M	Start Time	End Time
Tape: 1,	Task: 1	0.000778	2.111256
Tape: 2,	Task: 1	2.111290	4.222050
Tape: 3,	Task: 1	4.222069	6.332187
Tape: 1,	Task: 2	2.111284	4.264960
Tape: 2,	Task: 2	4.264969	6.418028
Tape: 3,	Task: 2	6.418047	8.571485
Tape: 1,	Task: 3	4.264968	6.500460
Tape: 2,	Task: 3	6.500470	8.735664
Tape: 3,	Task: 3	8.735684	10.971431
Tape: 1,	Task: 4	6.500467	8.565878
Tape: 2,	Task: 4	8.735681	10.800932
Tape: 3,	Task: 4	10.971438	13.037332
Tape: 1,	Task: 5	8.565884	10.627169
Tape: 2,	Task: 5	10.800939	12.862783
Tape: 3,	Task: 5	13.037335	15.098530
Tape: 1,	Task: 12	10.627177	12.377469
Tape: 2,	Task: 12	12.862786	14.612806
Tape: 3,	Task: 12	15.098542	16.849390

4.3 Вывод по экспериментальной части

Таким образом, исходя из полученных данных, можно сделать вывод, что при заданных условиях - время обработки на ленте больше времени генерации заданий, ограничение на вместимость очереди, задачи могут теряться и конвейер не гарантирует, что обработает все сгенерированные задачи, однако все задачи, поступившие на ленту, ганартированно будут

обработаны. Также, на основе полученного времени можно наблюдать параллельность обработки задач. Сгенерированные последовательно задачи обрабатываются тремя разными лентами, работающими параллельно, при этом время поступления на каждую ленту отличается на Δt примерно равное 28×10^{-6}, поскольку происходит работа с очередью. Если первая задача поступила на первую ленту в момент времени 0.000778с и завершила обработку в 2.111256с после чего покинула ленту, второй объект поступил на ленту 1 в 2.111284с.

Заключение

Таким образом, в ходе лабораторной работы было сделано следующее:

- изучена работа конвейера;
- описан процесс стадийной обработки данных;
- спроектирована структура ПО;
- сформулированы требования к ПО;
- описан механизм обмена данными;
- реализована конвейерная обработка задач.

Были также сделаны выводы на основе полученных из лога данных. При определенных условиях, конвейер не поддерживает обработку всех сгенерированных тасков, что осложняет контроль над лентами. Кроме того часть задач теряется. Однако, задачи, полученные из ленты всегда будут обработаны. Для того чтобы таски не терялись хотя при обработке между лентами, стоит выбирать алгоритмы время выполнения которых приблизительно равно.

Список литературы

[1] Уильямс Э., "Параллельное программирование на C++ в действии. Практика разработки многопоточных программ"

[2] Стандарт языка C++11 согласно ISO. [Электронный ресурс]. URL: https://isocpp.org/std/the-standard