МГТУ им. Баумана

Лабораторная работа №5

По курсу: "Анализ алгоритмов"

Вычислительный конвейер

Работу выполнила: Аминов Тимур, ИУ7-55Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Оглавление

Введение

Целью данной работы является получение навыка организации асинхронного взаимодействия между потоками на примере конвейерных вычислений.

Задачи лабораторной работы:

1) поставить задачу стадийной обработки данных;
2) спроектировать ПО, реализующее конвейерную обработку;
3) описать реализацию;
4) провести исследование времени обработки данных на основании журнала(лога);
5) интерпретировать данные лога.

Аналитическая часть

Конвейер — способ организации вычислений, используемый в современных процессорах и контроллерах с целью повышения их производительности (увеличения числа инструкций, выполняемых в единицу времени).

Один из самых простых и наиболее распространенных способов повышения быстродействия процессоров — конвейеризация процесса вычислений.

Конвейеризация – это техника, в результате которой задача или команда разбивается на некоторое число подзадач, которые выполняются последовательно. Каждая подкоманда выполняется на своем логическом устройстве. Все логические устройства (ступени) соединяются последовательно таким образом, что выход i-ой ступени связан с входом (i+1)-ой ступени, все ступени работают одновременно. Множество ступеней называется конвейером. Выигрыш во времени достигается при выполнении нескольких задач за счет параллельной работы ступеней, вовлекая на каждом такте новую задачу или команду.

Оценка производительности конвейера

Пусть задана операция, выполнение которой разбито на n последовательных этапов. При последовательном их выполнении операция выполняется за время

$$\tau_e = \sum_{i=1}^{n} \tau_i \tag{1}$$

где

n — количество последовательных этапов;

τ_i — время выполнения i-го этапа;

Быстродействие одного процессора, выполняющего только эту операцию, составит

$$S_e = \frac{1}{\tau_e} = \frac{1}{\sum\limits_{i=1}^{n} \tau_i} \tag{2}$$

где

τ_e — время выполнения одной операции;

n — количество последовательных этапов;

τ_i — время выполнения i-го этапа;

Выберем время такта — величину $t_T = max\sum_{i=1}^{n}(\tau_i)$ и потребуем при разбиении на этапы, чтобы для любого i = 1, ... , n выполнялось условие $(\tau_i + \tau_{i+1})modn = \tau_T$. То есть чтобы никакие два последовательных этапа (включая конец и новое начало операции) не могли быть выполнены за время одного такта.

Максимальное быстродействие процессора при полной загрузке конвейера составляет

$$S_{max} = \frac{1}{\tau_T} \tag{3}$$

где

τ_T — выбранное нами время такта;

Число n — количество уровней конвейера, или глубина перекрытия, так как каждый такт на конвейере параллельно выполняются n операций. Чем больше число уровней (станций), тем больший выигрыш в быстродействии может быть получен.

Известна оценка

$$\frac{n}{n/2} \leq \frac{S_{max}}{S_e} \leq n \tag{4}$$

где

S_{max} — максимальное быстродействие процессора при полной загрузке конвейера;

S_e — стандартное быстродействие процессора;

n — количество этапов.

то есть выигрыш в быстродействии получается от n/2 до n раз [2].

Реальный выигрыш в быстродействии оказывается всегда меньше, чем указанный выше, поскольку:

1) некоторые операции, например, над целыми, могут выполняться за меньшее количество этапов, чем другие арифметические операции. Тогда отдельные станции конвейера будут простаивать;

2) при выполнении некоторых операций на определённых этапах могут требоваться результаты более поздних, ещё не выполненных этапов предыдущих операций. Приходится приостанавливать конвейер;

3) поток команд(первая ступень) порождает недостаточное количество операций для полной загрузки конвейера.

Конструкторская часть

Разработка реализаций алгоритмов

Общая идея конвейера связана с разбиением некоторого процесса обработки объектов на независимые этапы и организацией параллельного выполнения во времени различных этапов обработки различных объектов, передвигающихся по конвейеру от одного этапа к другому. Поэтому основой разработки конвейера является разбиение процесса на независимые этапы.

Конвейер состоит из трех лент, которые называеются PreProcessing, Processing и PostProcessing. Каждый объект проходит три этапа обработки на каждой из лент. Объект представляет собой экземпляр специально созданного класса MyObject. Объекты класса MyObject по сути являются абрстракцией объектов, которые обрабатывались бы в реальной конкретной системе с конвейерными вычислениями. Три ленты конвейера представляют собой три отдельных класса, каждый из которых имеет функцию, выполняющую условно полезную работу, определяющую время нахождения объекта на ленте. Каждая лента запускается в отдельном потоке.

В программе N объектов генерируются и помещаются в очередь первой ленты (PreProcessing). После того, как i-й объект (i = 1, ..., N) был обработан на первой ленте, он передается в очередь второй ленты (Processing). После обработки на второй ленте объект передается в очередь третьей ленты (PostProcessing). После обработки на третьей ленты объект помещается в контейнер обработанных объектов. Объект считается обработанным, если он прошел все три ленты конвейера. Эти действия выполняются для каждого из N сгенерированных объектов. Для каждой ленты были определены следующие функции, выполняющие условно полезную работу: вычисление факториала числа, вычисление наибольшего общего делителя, помноженного на константу для числа и константы, возведение числа в заранее заданную степень для PreProcessing, Processing и PostProcessing соответственно. Класс MyObject имеет целочисленное поле a. Данное число является условно полезной информацией, которая и преобразуется в конвейере путем поочередной подачи числа a на вход в функцию каждой ленты.

На рисунке 1 изображена схема алгоритма обработки объектов класса MyObject. Принцип обработки объектов на ленте PreProcessing изображен на рисунке 2. Принцип обработки объектов на ленте Processing изображен на рисунке 3. Принцип обработки объектов на ленте PostProcessing изображен на рисунке 4.

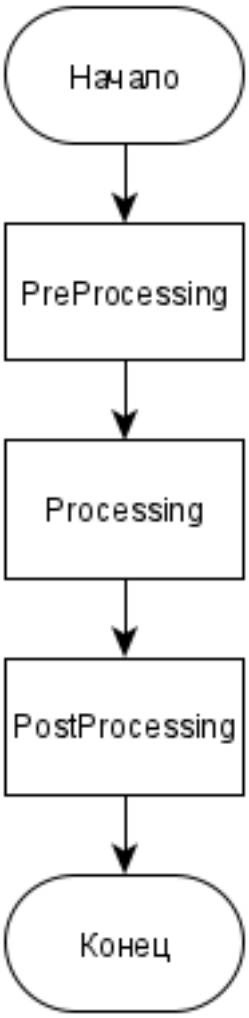

Рис. 1: Алгоритм обработки объектов класса MyObject

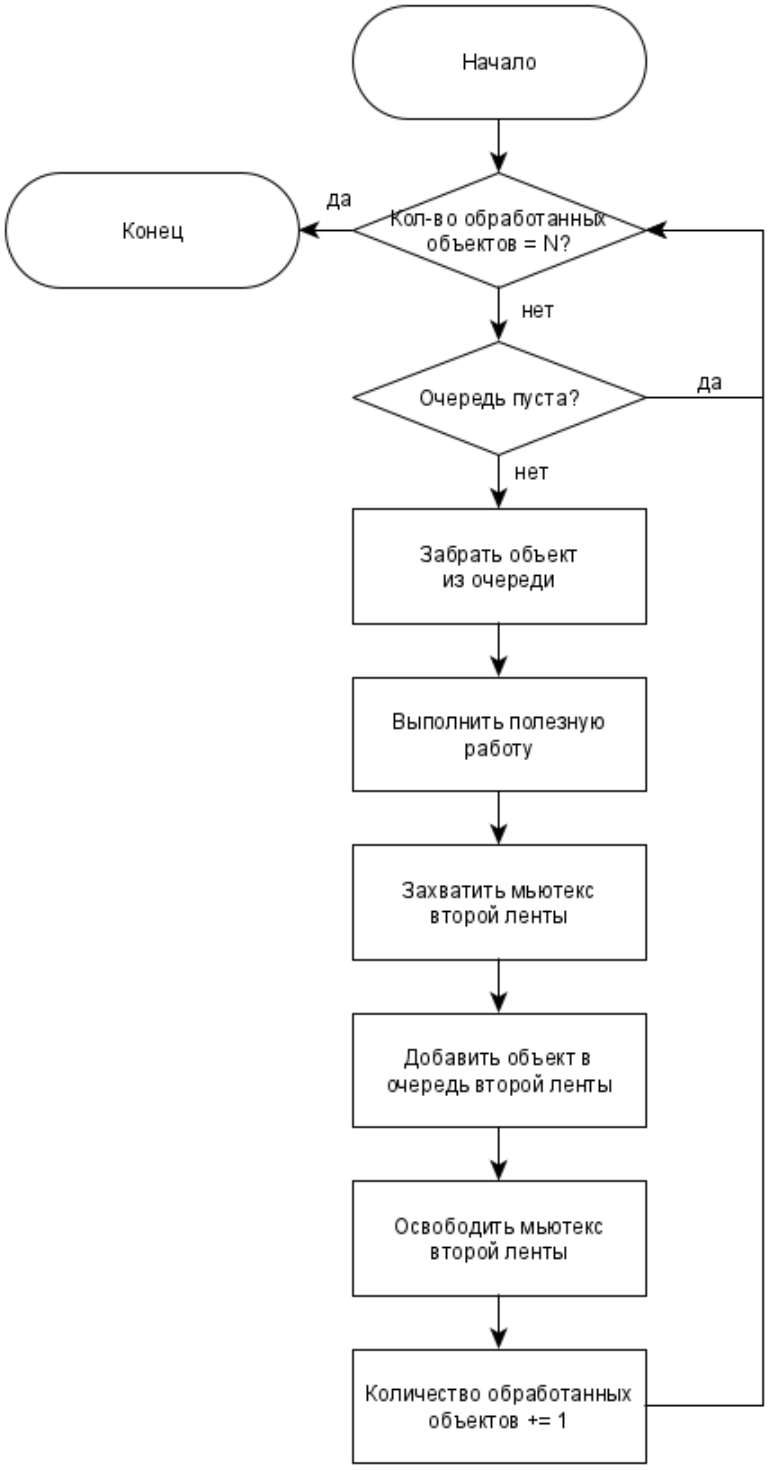

Рис. 2: Алгоритм обработки объектов на ленте PreProcessing

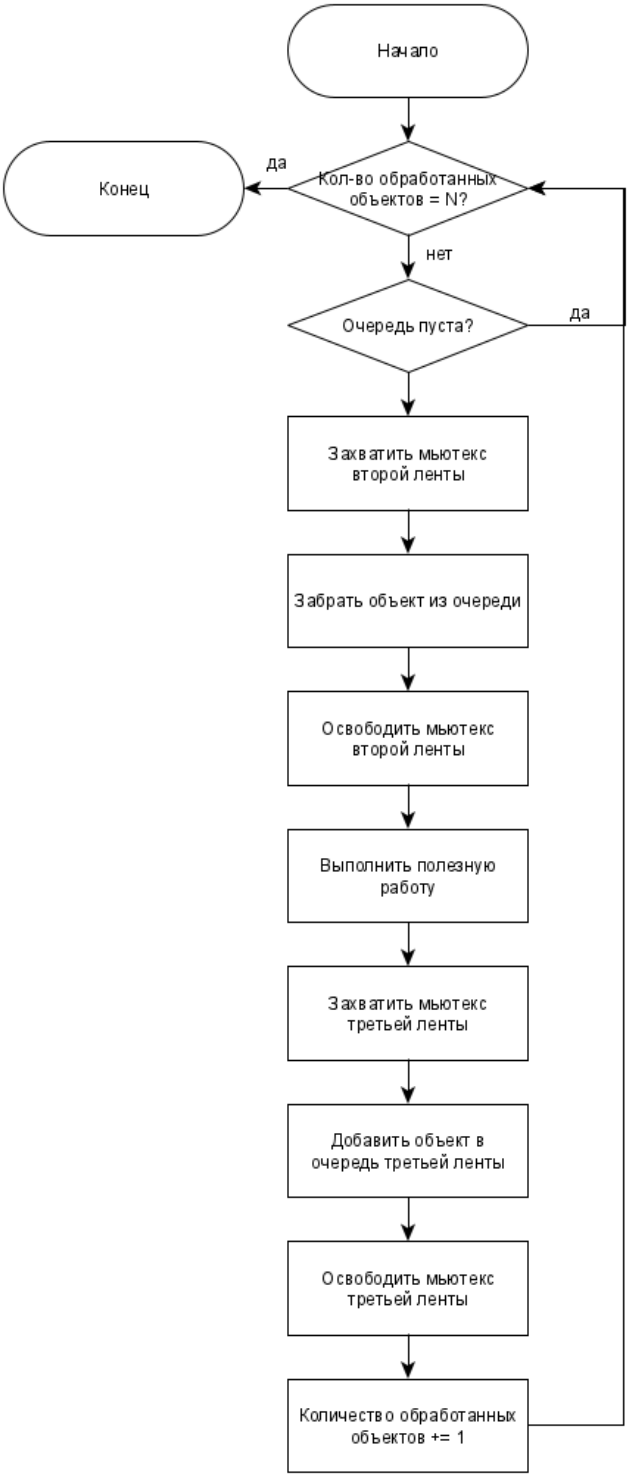

Рис. 3: Алгоритм обработки объектов на ленте Processing

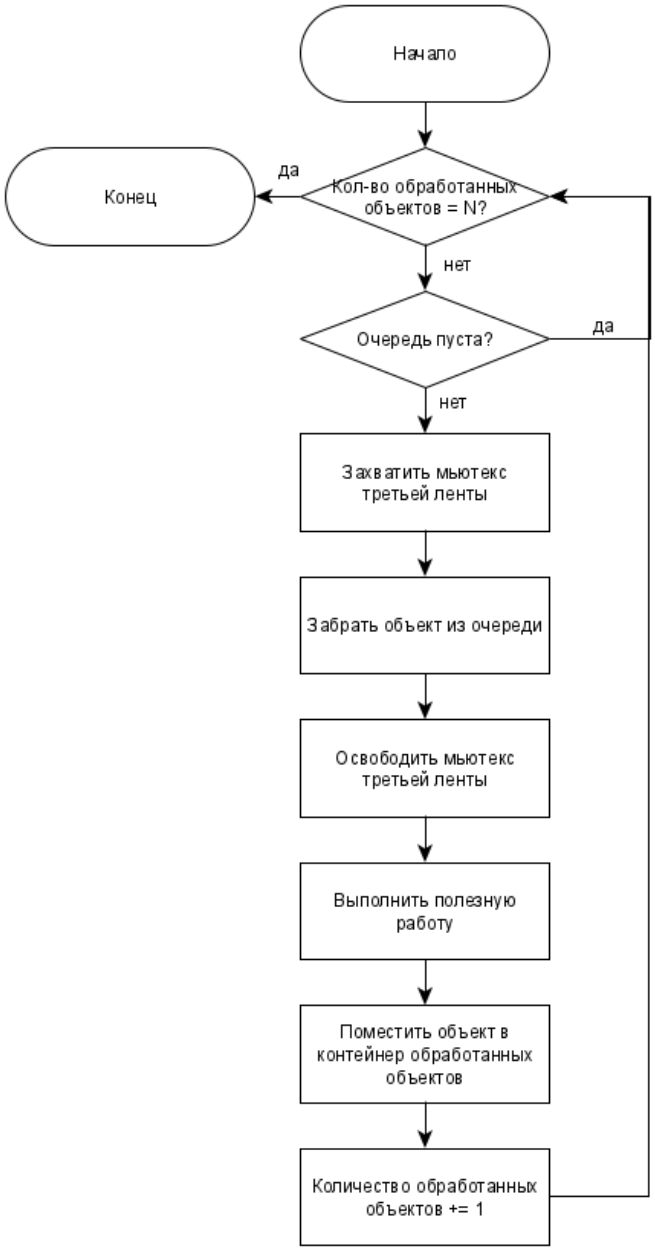

Рис. 4: Алгоритм обработки объектов на ленте PostProcessing

Требования к программе

Требования к вводу:
Программа не имеет входных данных.
Требования к программе:
На выходе программа предоставляет лог-файл, где для каждого обработанного на конвейете объекта записано:

1. Время начала обработки на первой ленте и время конца обработки на первой ленте;

2. Время начала обработки на второй ленте и время конца обработки на второй ленте;

3. Время начала обработки на третьей ленте и время конца обработки на третьей ленте.

Кроме этого, в лог-файл долнжо быть записано общее время работы конвейера.

Технологическая часть

В данном разделе будут определены средства реализации и приведен листинг кода.

Выбор языка программирования

В качестве языка программирования для реализации программы был выбран язык C++ и фреймворк Qt, потому что:

- язык C++ имеет высокую вычислительную производительность;
- язык C++ поддерживает различные стили программирования;
- в Qt существует удобный инструмент для тестирования - QtTest - который позволяет собирать тесты в группы, собирать результаты выполнения тестов, а также уменьшить дублирование кода при схожих объектах тестирования.

Для замеров времени использовался методы restart() и elapsed() класса QTime. Метод elapsed() возвращает количество миллисекунд, прошедших с момента последнего вызова start() или restart().

Для работы с мьютексами и потоками в лабораторной работе используются классы QMutex, QThread фреймворка Qt.

Для реализации очереди использовался класс queue из стандартной библиотеки C++.

Сведения о модулях программы

Программа состоит из следующих файлов:

- myobject.h, myobject.cpp - заголовочный файл и файл, в котором расположена реализация обрабатываемых объектов;
- main.cpp - главный файл программы, в котором расположен запуск конвейера;
- preprocessing.h, preprocessing.cpp - файл и заголовочный файл, в котором расположена реализация первой ленты конвейера;

- processing.h, processing.cpp - файл и заголовочный файл, в котором расположена реализация второй ленты конвейера;
- postprocessing.h, postprocessing.cpp - файл и заголовочный файл, в котором расположена реализация третьей ленты конвейера.

Листинги кода алгоритмов

Листинг 1: Запуск конвейера

```
const int COUNT = 10;

const string LOG_FILE = "times.txt";
const string LOG_FILE2 = "times2.txt";

void consistent(MyObject **objs);

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    srand(static_cast<unsigned int>(time(nullptr)));

    MyObject *objs[COUNT];

    for (int i = 0; i < COUNT; ++i)
    {
        int myrand = rand() % MAX_RAND + 10;

        objs[i] = new MyObject(i, myrand);
    }

    consistent(objs);

    QTime timer;
    timer.restart();

    int start_time = timer.elapsed();

    vector<MyObject> dump;

    QMutex *mutex2 = new QMutex;
    QMutex *mutex3 = new QMutex;

    PostProcessing *postproc = new PostProcessing(COUNT, &timer, &dump
        , mutex2);
    Processing *proc = new Processing(COUNT, &timer, postproc, mutex2,
        mutex3);
    PreProcessing *preproc = new PreProcessing(COUNT, &timer, proc,
       mutex3);

    for (int i = 0; i < COUNT; ++i)
    {
```

```
40          preproc->addToQueue(*objs[i]);
41      }
42
43      vector<thread> threads;
44      threads.push_back(thread(&PreProcessing::process, preproc));
45      threads.push_back(thread(&Processing::process, proc));
46      threads.push_back(thread(&PostProcessing::process, postproc));
47
48      for (unsigned int i = 0; i <  threads.size(); ++i)
49      {
50          if (threads.at(i).joinable())
51              threads.at(i).join();
52      }
53
54      int total_time = timer.elapsed() - start_time;
55
56      ofstream fout(LOG_FILE);
57      if (fout.is_open())
58      {
59          for (unsigned int i = 0; i < dump.size(); ++i)
60              dump.at(i).timesToFile(fout);
61          fout << "Total time: " << total_time << endl;
62          cout << "Results in: " << LOG_FILE << endl;
63      }
64      else
65          cout << "I cant open file " << LOG_FILE << endl;
66      fout.close();
67
68
69      delete mutex2;
70      delete mutex3;
71      delete preproc;
72      delete proc;
73      delete postproc;
74
75      return 0;
76  }
```

Листинг 2: Объявление класса MyObject

```
1  class MyObject
2  {
3  public:
4      MyObject(int id, int a);
5      void setTime(int time);
6      void printTimes();
7      void timesToFile(ofstream &fout);
8      void set_a(unsigned long int a) {this->a = a;}
9      unsigned long int get_a() {return  this->a;}
10
11 private:
12     unsigned long int a;
13     int _id;
14     vector<int> _times;
```

```
};
```

Листинг 3: Реализация класса MyObject

```
MyObject::MyObject(int id)
{
    this->_id = id;
}

void MyObject::setTime(int time)
{
    this->_times.push_back(time);
}

void MyObject::printTimes()
{
    cout << "Object" << _id << "\t\t";
    for (unsigned int i = 0; i < _times.size(); ++i)
        cout << _times.at(i) << "  ";
    cout << endl;
}

void MyObject::timesToFile(ofstream &fout)
{
    fout << "Object" << _id << "\t\t";
    for (unsigned int i = 0; i < _times.size(); ++i)
    {
        fout << _times.at(i) << "  ";
    }
    fout << endl;
}
```

Листинг 4: Объявление класса PreProcessing

```
class PreProcessing
{
public:
    PreProcessing(int count, QTime *timer, Processing *p, QMutex *
       mutex2);
    void addToQueue(MyObject obj);
    void process();
private:
    queue<MyObject> _queue;
    QTime *_timer;
    Processing *_proc;
    QMutex *_mutex2;
    int _count;
    int preprocessed = 0;
    unsigned long int fac(unsigned long int num);
};
```

Листинг 5: Реализация класса PreProcessing

```
PreProcessing::PreProcessing(int count, QTime *timer, Processing *p,
   QMutex *mutex2)
```

```
{
    this->_count = count;
    this->_timer = timer;
    this->_proc = p;
    this->_mutex2 = mutex2;
}

void PreProcessing::addToQueue(MyObject obj)
{
    _queue.push(obj);
}

void PreProcessing::process()
{
    while (preprocessed != _count)
    {
        if (_queue.size() != 0)
        {
            // cout << "PreProcessing" << _timer->elapsed() << endl;
            MyObject obj = _queue.front();

            obj.setTime(_timer->elapsed());

            unsigned long fac = this->fac(obj.get_a());
            obj.set_a(fac);

            obj.setTime(_timer->elapsed());
            _queue.pop();
            _mutex2->lock();
            _proc->addToQueue(obj);
            _mutex2->unlock();
            preprocessed++;
        }
    }
}

unsigned long PreProcessing::fac(unsigned long int num)
{
    unsigned long int fac = 1;
    for (unsigned long int count = 1; count <= num; count++)
    {
        fac *= count;
    }

    return  fac;
}
```

Листинг 6: Объявление класса Processing

```
class Processing
{
public:
    Processing(int count, QTime *timer, PostProcessing *p, QMutex *
       mutex2, QMutex *mutex3);
```

```
5       void addToQueue(MyObject obj);
6       void process();
7   private:
8       queue<MyObject> _queue;
9       QTime *_timer;
10      PostProcessing *_proc;
11      QMutex *_mutex2;
12      QMutex *_mutex3;
13      int _count;
14      int processed = 0;
15      unsigned long int nod_pow(unsigned long int a, unsigned long int b
            );
16  };
```

Листинг 7: Реализация класса Processing

```
1   Processing::Processing(int count, QTime *timer, PostProcessing *p,
        QMutex *mutex2, QMutex *mutex3)
2   {
3       this->_count = count;
4       this->_timer = timer;
5       this->_proc = p;
6       this->_mutex2 = mutex2;
7       this->_mutex3 = mutex3;
8   }
9
10  void Processing::addToQueue(MyObject obj)
11  {
12      _queue.push(obj);
13  }
14
15  void Processing::process()
16  {
17      while (processed != _count)
18      {
19          if (_queue.size() != 0)
20          {
21              // cout << "Processing" << _timer->elapsed() << endl;
22              MyObject obj = _queue.front();
23              QThread thread;
24              obj.setTime(_timer->elapsed());
25
26              unsigned long nod_pow = this->nod_pow(obj.get_a(), 357);
27              obj.set_a(nod_pow);
28
29              obj.setTime(_timer->elapsed());
30
31              _mutex2->lock();
32              _queue.pop();
33              _mutex2->unlock();
34
35              _mutex3->lock();
36              _proc->addToQueue(obj);
37              _mutex3->unlock();
```

```
38
39              processed++;
40          }
41      }
42  }
43
44  unsigned long Processing::nod_pow(unsigned long a, unsigned long b)
45  {
46      unsigned long int nod = 0;
47
48      for (unsigned long int i = a; i > 0; i--)
49      {
50          if (a % i == 0 && b % i == 0)
51          {
52              nod = i;
53              break;
54          }
55      }
56
57      unsigned long result = 1;
58
59      for (int i = 0; i < 10000000; i++)
60          result *= nod;
61
62      return  result;
63  }
```

Листинг 8: Объявление класса PostProcessing

```
1  class PostProcessing
2  {
3  public:
4      PostProcessing(int count, QTime *timer, vector<MyObject> *dump,
           QMutex *mutex3);
5      void addToQueue(MyObject obj);
6      void process();
7  private:
8      queue<MyObject> _queue;
9      QTime *_timer;
10     vector<MyObject> *_dump;
11     QMutex *_mutex3;
12     int _count;
13     int postprocessed = 0;
14     unsigned long int power(unsigned long int a, unsigned long int b);
15 };
```

Листинг 9: Реализация класса PostProcessing

```
1  PostProcessing::PostProcessing(int count, QTime *timer, vector<
       MyObject> *dump, QMutex *mutex3)
2  {
3      this->_count = count;
4      this->_timer = timer;
5      this->_dump = dump;
6      this->_mutex3 = mutex3;
```

```
7  }
8
9  void PostProcessing::addToQueue(MyObject obj)
10 {
11     _queue.push(obj);
12 }
13
14 void PostProcessing::process()
15 {
16     while (postprocessed != _count)
17     {
18         if (_queue.size() != 0)
19         {
20             // cout << "PostProcessing" << _timer->elapsed() << endl;
21             MyObject obj = _queue.front();
22             QThread thread;
23             obj.setTime(_timer->elapsed());
24
25             unsigned long int power = this->power(obj.get_a(),
                   1000000000);
26             obj.set_a(power);
27
28             obj.setTime(_timer->elapsed());
29
30             _mutex3->lock();
31             _queue.pop();
32             _mutex3->unlock();
33
34             _dump->push_back(obj);
35             postprocessed++;
36         }
37     }
38 }
39
40 unsigned long PostProcessing::power(unsigned long a, unsigned long b)
41 {
42     unsigned long int res = 1;
43
44     for (unsigned long int i = 0; i < b; i++)
45     {
46         res *= a;
47     }
48
49     return  res;
50 }
```

Тесты

Тестирование ПО проводилось вручную на основании данных лог-файла. Проверялась работа и корректное завершение программы для разного количества объектов: для 1 - 5 объектов, от 10 до 100 с шагом 10. Все тесты были

пройдены.

Экспериментальная часть

В данном разделе будет сравнительный анализ реализаций конвейера при разной нагруженности компонентов конвейера.

Постановка эксперимента

В рамках данной работы были проведены следующие эксперименты.

1. Измерение времени работы конвейера в три потока при его загруженности от 0 до 20 элементами с шагом 15.

2. Измерение времени работы конвейера в один поток (последовательно) при его загруженности от 0 до 25 элементами с шагом 5.

Сравнительный анализ на материале экспериментальных данных

Результаты замеров времени работы конвейера в три потока отображены в таблице 1. Результаты замеров времени работы конвейера в один поток отображены в таблице 2.

Таблица 1: Результаты замеров времени для трех потоков

Кол-во объектов	время, мс
0	0
10	499
20	601
30	1039
40	1369
50	1628

Таблица 2: Результаты замеров времени для одного потока

Кол-во объектов	время, мс
0	0
10	917
20	1262
30	2012
40	2535
50	3036

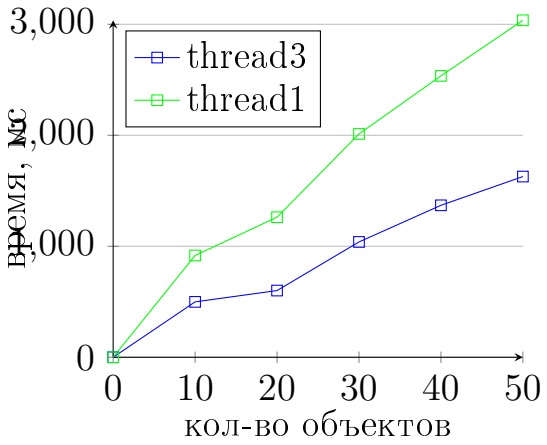

Рис. 5: Результаты замеров времени для конвейера в три потока и в один поток

Выводы

В результате проведенных экспериментов было установлено следующее.

1. Время работы конвейера в три потока практически линейно зависит от количества обрабатываемых объектов.

2. Время обработки объектов в один поток практически линейно зависит от количества обрабатываемых объектов.

3. Время обработки объектов в один поток растет быстрее времени работы конвейера в три потока при росте количества обрабатываемых объектов.

Лог-файл и его интерпретация

Ниже представлены данные из лог-файл, который создает программа.
Условные обозначения:

1. t11, t12 - время поступления объекта на первую ленту и время конца его обработки на первой ленте соответсвенно в мс;

2. t21, t22 - время поступления объекта на вторую ленту и время конца его обработки на первой ленте соответсвенно в мс;

3. t31, t32 - время поступления объекта на третью ленту и время конца его обработки на первой ленте соответсвенно в мс;

4. id объекта - его номер в очереди.

Таблица 3: Данные из лог-файла

ID объекта	t11, мс	t12, мс	t21, мс	t22, мс	t31, мс	t32, мс
0	0	110	110	125	125	156
1	110	203	203	235	235	250
2	203	313	313	328	328	360
3	313	406	406	438	438	453
4	406	500	500	516	516	547
5	500	605	605	627	627	649
6	605	703	703	723	723	739
7	703	802	802	817	817	833
8	802	912	912	932	932	954
9	912	1014	1014	1036	1036	1057

Total time: 1057

Из данных лог-файла можно сделать вывод, что конвеер работает корректно. Общее время работы конвейера для 10 объектов составило 1057 мс.

Заключение

В ходе данной лабораторной работы были получены навыки огранизации асинхронного взаимодействия между потоками на примере конвейерных вычислений. Было спроектировано ПО, реализующее конвейерную обработку. Был проведен сравнительный анализ времени работы конвейера при работе в один поток и в три. Были проанализированы данные из лог-файла.

Литература

[1] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход.- М.:Техносфера, 2009.

[2] Д. Кнут. Искусство программирования, М., Мир, 1978

[3] Воеводин В.В. Математические модели и методы в параллельных процессах. М., 1986

[4] Погорелов Д.А. Применение конвейерной обработки данных на примере сортировки простыми вставками, М., Образование и наука в России и за рубежом 2019 .- Т. 49 , № 1

[5] Корнеев В.В. Параллельные вычислительные системы. М., 1999.

[6] ISO/IEC 14882:2017 [Электронный ресурс]. – Режим доступа: https://www.iso.org/standard/68564.html, свободный – (27.11.2019)

[7] <chrono> [Электронный ресурс]. – Режим доступа: http://www.cplusplus.com/reference/chrono/, свободный – (20.11.2019)