

Министерство образования Российской Федерации
Московский Государственный Технический
Университет им. Н.Э. Баумана

Отчет по лабораторной работе №6
По курсу «Анализ алгоритмов»

**Тема: «Конвейерная обработка
данных»**

Студент: **Горохова И.Б.**
Группа: **ИУ7-51**

Преподаватель: **Волкова Л.Л.**

Москва, 2017

Содержание

Постановка задачи	3
Реализация	3
Реализация на языке программирования	3
Этапы обработки	9
Пример работы программы	10
Устройство конвейера	10
Эксперимент	11
Заключение	12

Постановка задачи

В ходе лабораторной работы предстоит:

1. реализовать конвейерную обработку данных с помощью потоков;
2. сравнить реализованный конвейер с однопоточной обработкой тех же данных.

Реализация

Для реализации конвейерной обработки с помощью потоков было выделено три этапа, на каждом из которых производится своя операция обработки данных. В первом потоке выполняется первая операция, после чего выходные данные передаются во второй поток для выполнения второй операции. Выходные данные после выполнения третьей операции подаются в третий поток для выполнения третьей операции.

В качестве входных данных выбраны строки, в качестве операций - преобразования над строками.

Для реализации на языке Java были написаны три класса с интерфейсами **Runnable**, каждый из которых в своем потоке выполняет свою операцию, после чего передает данные на следующий этап.

Реализация классов схожа. Однако в класс **First** на вход подается очередь из исходных данных, а в классы **Second** и **Third** на вход подаются пустые очереди. Для очереди использован класс **ConcurrentLinkedQueue**. В листингах 1-3 приведена реализация классов **First** для первого потока, **Second** для второго потока и **Third** для третьего потока.

Листинг 1: Класс **First**

```
1 class First implements Runnable {  
2     boolean cancel = false;  
3     private ConcurrentLinkedQueue<String> queue;  
4     private Second next;  
5  
6     First(ConcurrentLinkedQueue<String> q) {  
7         queue = q;  
8     }  
9  
10    @Override
```

```

11     public void run() {
12         while (!cancel) {
13             String data = queue.poll();
14             if (data != null) {
15                 System.out.println("0: " + data);
16                 data = addInverse(data);
17                 next.enqueue(data);
18                 System.out.println("1: "+ data);
19             }
20             else {
21                 cancel();
22             }
23         }
24     }
25
26     private void cancel() {
27         cancel = true;
28         System.out.println("1: FINISHED");
29     }
30
31     void setNext(Second next) {
32         this.next = next;
33     }
34
35     private String addInverse(String str) {
36         StringBuilder res = new StringBuilder(str);
37         for (int i = str.length()-1; i >= 0; i--)
38             res.append(str.charAt(i));
39         return res.toString();
40     }
41 }

```

Поля класса First:

- `cancel` - флаг окончания работы потока, который поставится в `true`, когда очередь из входных данных опустеет;
- `queue` - очередь, из которой берутся данные (строки) для обработки;
- `next` - ссылка на следующий этап.

Методы класса First:

- `run()` - содержит код для потока. Пока очередь не пуста, с помощью `queue.poll()` из очереди берутся данные, производится операция над

ними, далее данные добавляются в очередь ко второму потоку с помощью *next.enqueue(data)*;

- *cancel()* - ставит флаг *cancel* в *true*;
- *setNext()* - добавляет в *next* ссылку на следующий этап;
- *addInverse()* - функция преобразования строки на данном этапе.

Листинг 2: Класс Second

```
1 class Second implements Runnable {
2     boolean cancel = false;
3     private ConcurrentLinkedQueue<String> queue = new
        ConcurrentLinkedQueue<String>();
4     private First prev;
5     private Third next;
6
7     public void run() {
8         while (!cancel) {
9             String data = queue.poll();
10            if (data != null) {
11                data = GetEvenLiterals(data);
12                next.enqueue(data);
13                System.out.println("    2: " + data);
14
15            }
16            else {
17                cancel();
18            }
19        }
20    }
21
22    void enqueue(String data) {
23        queue.offer(data);
24    }
25
26    private void cancel() {
27        if (prev.cancel) {
28            cancel = true;
29            System.out.println("    2: FINISHED");
30        }
31    }
32 }
```

```

33     void setPrev(First prev) {
34         this.prev = prev;
35     }
36
37     void setNext(Third next) {
38         this.next = next;
39     }
40
41     private String GetEvenLiterals(String str) {
42         StringBuilder res = new StringBuilder();
43         for (int i = 0; i < str.length(); i+=2) {
44             res.append(str.charAt(i));
45         }
46         return res.toString();
47     }
48 }
49
50 }

```

Поля класса Second:

- cancel - флаг окончания работы потока, который поставится в true, когда очередь из входных данных опустеет;
- queue - очередь, из которой берутся данные (строки) для обработки;
- next - ссылка на следующий этап;
- prev - ссылка на предыдущий этап.

Методы класса Second:

- run() - содержит код для потока. Пока очередь не пуста, с помощью *queue.poll()* из очереди берутся данные, производится операция над ними, далее данные добавляются в очередь к третьему потоку с помощью *next.enqueue(data)*;
- enqueue(data) - добавляет data в очередь queue;
- cancel() - ставит флаг cancel в true;
- setNext() - добавляет в next ссылку на следующий этап;
- setPrev() - добавляет в prev ссылку на предыдущий этап;

- `getEvenLiterals()` - функция преобразования строки на данном этапе.

Листинг 3: Класс Third

```
1 class Third implements Runnable {
2     private boolean cancel = false;
3     private ConcurrentLinkedQueue<String> queue = new
        ConcurrentLinkedQueue<String>();
4     private Second prev;
5
6     public void run() {
7         while (!cancel) {
8             String data = queue.poll();
9             if (data != null) {
10                 data = chooseLiterals(data);
11                 System.out.println("        3: " + data);
12
13             }
14             else {
15                 cancel();
16             }
17         }
18     }
19
20     void enqueue(String data) {
21         queue.offer(data);
22     }
23
24     private void cancel() {
25         if (prev.cancel) {
26             cancel = true;
27             System.out.println("        3: FINISHED");
28         }
29     }
30
31     void setPrev(Second prev) {
32         this.prev = prev;
33     }
34
35     private String chooseLiterals(String str) {
36         StringBuilder res = new StringBuilder();
37         int len = str.length();
38         for (int i = 0; i < len/2; i++) {
```

```

39         res.append(str.charAt(i));
40         res.append(str.charAt(len - i - 1));
41     }
42     if (len % 2 == 1)
43         res.append(str.charAt((int)(len/2)));
44     return res.toString();
45 }
46
47 }

```

Поля класса Third:

- cancel - флаг окончания работы потока, который поставится в true, когда очередь из входных данных опустеет;
- queue - очередь, из которой берутся данные (строки) для обработки;
- prev - ссылка на предыдущий этап.

Методы класса Third:

- run() - содержит код для потока. Пока очередь не пуста, с помощью *queue.poll()* из очереди берутся данные, производится операция над ними; *next.enqueue(data)*;
- enqueue(data) - добавляет data в очередь queue;
- cancel() - ставит флаг cancel в true;
- setPrev() - добавляет в prev ссылку на предыдущий этап;
- chooseLiterals() - функция преобразования строки на данном этапе.

Количество этапов (потоков) не ограничено и увеличивается путём добавления нового класса со схожей реализацией.

Данные и операции над ними так же могут быть любыми.

В листинге 4 представлен код функции main:

```

1    ConcurrentLinkedQueue<String> queue = new
      ConcurrentLinkedQueue<String>();
2    FillQueue(queue);
3    First step1 = new First(queue);
4    Second step2 = new Second();
5    Third step3 = new Third();
6    step1.setNext(step2);

```



```

7      step2.setPrev(step1);
8      step2.setNext(step3);
9      step3.setPrev(step2);
10     Thread th1 = new Thread(step1);
11     Thread th2 = new Thread(step2);
12     Thread th3 = new Thread(step3);
13     th1.start();
14     th2.start();
15     th3.start();
16     try {
17         th1.join();
18         th2.join();
19         th3.join();
20     } catch (InterruptedException e) {
21         e.printStackTrace();
22     }

```

Строки 1-2: создание и наполнение очереди исходными данными;
 Строки 3-9: инициализация этапов First, Second, Third;
 Строки 10-15: создание и запуск потоков для этапов First, Second, Third;
 Строки 16-22: ожидание окончания работы потоков.

Этапы обработки строки

На первом этапе, выполняющемся в первом потоке, проводится конкатенация исходной строки с инверсивной исходной строкой. На втором этапе, выполняющемся на втором потоке, производится выборка букв на четных позициях и получение из них новой строки. На третьем этапе, выполняющемся на третьем потоке, производится выборка букв в порядке первая-последняя-вторая-предпоследняя... и образование из них новой строки. Пример работы представлен в Таблице 1.

Таблица 1. Преобразования строк.

Входная строка	1 этап	2 этап	3 этап
слово	словооволс	соовл	слово
буква	букваавкуб	бкаву	буква
символ	символловмис	смолви	символ

Пример работы программы

0: мишень	//строка1
1: мишенььнешим	//строка1 этап1
0: пуля	//строка2
1: пуляялуп	//строка2 этап1
2: мшньей	//строка1 этап2
3: мишень	//строка1 этап3
0: соревнование	//строка3
1: соревнованиееинавонверос	//строка3 этап1
0: алгоритм	//строка4
1: алгоритммтирогла	//строка4 этап1
2: пляу	//строка2 этап2
3: пуля	//строка2 этап3
0: коррозия	//строка5
1: коррозияязоррок	//строка5 этап1
2: срвоаиенвнео	//строка3 этап2
3: соревнование	//строка3 этап3
2: агртмиол	//строка4 этап2
3: алгоритм	//строка4 этап3
1: FINISHED	
2: кроиязро	//строка5 этап2
3: коррозия	//строка5 этап3
3: FINISHED	
2: FINISHED	

Конвейер

На нижней части изображения 1 показан принцип работы трехэтапного конвейера. Идея этого конвейера заключается в параллельном выполнении операции в трех потоках: Как только первый поток заканчивает работать с данными, он отдает их в следующий поток, а сам продвигает свою работу с новыми данными. В это же время во втором потоке обрабатываются данные, полученные из первого потока. Аналогично выполняется передача данных и в третий поток.

На верхней части изображения 1 показано распределение времени на работу с каждой строкой при условии, что три операции будет выполнять один поток (без использования распараллеливания).

I,II,III	str1	str1	str1	str2	str2	str2	str3	str3	str3	str4	str4	str4
0												
III			str1	str2	str3	str4						
II		str1	str2	str3	str4							
I	str1	str2	str3	str4								
0	TIME											

Изображение 1. Конвейер на трех потоках.

Если принять, что каждая операция выполняется за время t , то для выполнения операций над N строками с помощью одного потока требуется

$F(N) = N * 3t$, а для выполнения операций над N строками с помощью конвейера потребуется

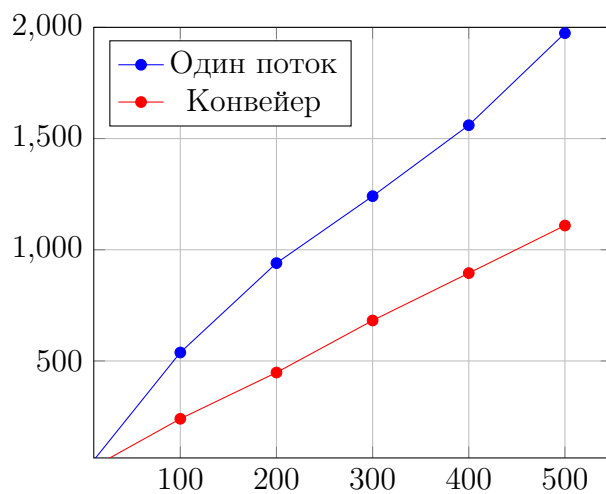
$$F(N) = (N + 2)t$$

Эксперимент

В качестве эксперимента были произведены замеры времени работы конвейера на очереди из 100, 200, 300, 400 и 500 строк. Также было замерено время на обработку очередей из этих же строк с помощью одного потока, выполняющего три этапа обработки последовательно. Результаты замеров приведены в таблице 2 и на графике 1 (по оси абсцисс - количество строк, по оси ординат - время в миллисекундах).

Таблица 2. Результаты замеров времени.

Количество строк в очереди	Время работы в одном потоке (мсек)	Время работы конвейера (мсек)
100	538	240
200	940	448
300	1241	682
400	1560	895
500	1974	1109



Изображение 2. Временной график.

Выводы из эксперимента

В результате эксперимента была подтверждена эффективность использования конвейера перед однопоточной обработкой данных. В результате теоретической оценки было выведено, что использование конвейера на трех потоках при больших входных данных должно давать преимущество в три раза, однако экспериментальные замеры показали, что конвейер эффективнее примерно в 2 раза. Это происходит из-за того, что на каждом этапе обработка данных занимает неодинаковое время.

Заключение

В ходе лабораторной работы я реализовала конвейерную обработку данных с помощью потоков на языке программирования Java и сравнила реализованный конвейер с однопоточной обработкой тех же данных.