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: **Медведев А.В.**
Группа: **ИУ7-51**

Преподаватель: **Волкова Л.Л.**

Москва, 2017

Содержание

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 C# был написан базовый класс от которого путем переопределения одного из методов создается очередной этап обработки.

Листинг 1: Класс ConveyerBase

```
internal abstract class ConveyerBase<T>
{

    private bool _cancel;
    protected bool Wait;
    private readonly Queue<T> _queue;
    private   onveyerBase  <T> _next;
    private readonly object _locker=new object();
    protected string Name="";

    protected ConveyerBase()
    {
        _queue= new Queue<T>();
        _cancel = false;
        Wait = true;
```

```
16      }
17      public void SetNext onveyer(ConveyerBase<T> next)
18      {
19          lock (_locker)
20          {
21              _next = next;
22          }
23      }
24      public void Run()
25      {
26
27
28
29          Console.WriteLine("{Name} :: Started");
30          while (!_cancel)
31          {
32              T data;
33
34              if (_queue.Count > 0)
35              {
36                  lock (_locker)
37                  {
38                      data = _queue.Dequeue();
39                  }
40              }
41              else
42              {
43                  if (Wait==false)
44                  {
45                      Cancel();
46                  }
47                  continue;
48              }
49              Console.WriteLine("{Name} <== {data}");
50              data = Work(data);
51              AddToNext(data);
52              Console.WriteLine($"{Name} ==> {data}");
53
54          }
55      }
56
57      public void Enqueue(T data)
58      {
```

```
59          lock (_locker)
60          {
61              _queue.Enqueue(data);
62          }
63      }
64
65      private void AddToNext(T data)
66      {
67          _next?.Enqueue(data);
68      }
69
70      private void Cancel()
71      {
72          _cancel = true;
73          _queue.Clear();
74          _next?.CancelNextWait();
75          Console.WriteLine("{Name} :: Finished");
76      }
77
78      private void CancelNextWait()
79      {
80          Wait = false;
81      }
82      protected abstract T Work(T data);
83  }
```

Поля класса ConveyerBase:

- cancel - Флаг отмены;
- Wait - Флаг необходимости ожидания прошлого этапа;
- queue - Очередь с данными
- next - Следующий этап
- locker - Блокировщик
- Name - Название потока
- next - ссылка на следующий этап.

Методы класса ConveyerBase:

- ConveyerBase() - Конструктор

- SetNextConveyer(ConveyerBase<T> next) - Установка следующего этапа конвеера
- Run() - Запуск выполнения работы;
- Enqueue(T data) - Добавление данных в очередь для обработки
- AddToNext(T data) - Передача данных в следущий этап обработки;
- Cancel() - Остановка обработки;
- CancelNextWait() - Отменена ожидания завершения
- abstract T Work(T data); - Процедура обрабатывающая данные

В листинге 2 приведена реализация классов выполняющих определенную работу с данными по принципу конвеера

Листинг 2: Класс First

```
class First:
 ConveyerBase<string>
   {
        public First()
        {
            Name = "1";
            Wait = false;
        }
        protected override string Work(string data)
        {
            int len = data.Length;
            for (int i = len-1; i >=0; i--)
            {
                data += data[i];
            }
            return data;

        }
   }

   class Second : ConveyerBase<string>
   {
        public Second()
        {
            Name = "2";
        }
```

```
27
28          protected override string Work(string data)
29          {
30              return data.Substring(0, data.Length/2) + "****
                    ";
31          }
32      }
33
34      class Third : ConveyerBase<string>
35      {
36          public Third()
37          {
38              Name = "3";
39          }
40
41
42
43          protected override string Work(string data)
44          {
45              string line="";
46              for (int i = 0; i < data.Length; i+=2)
47              {
48                  line += data[i];
49              }
50              return line;
51
52          }
53      }
```

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 данном случае класс First дописывает к исходной строке строку в обратном порядке. Второй класс к первой половине строки дописывает "****". А третий класс берет каждый второй символ из подаваемой ему строки.

В листинге 3 представлен код функции запускающий работу конвеера:

Листинг 3: Запуск конвеера

```
1 private void ConveyerProcessing(IEnumerable<string> content
    )
```

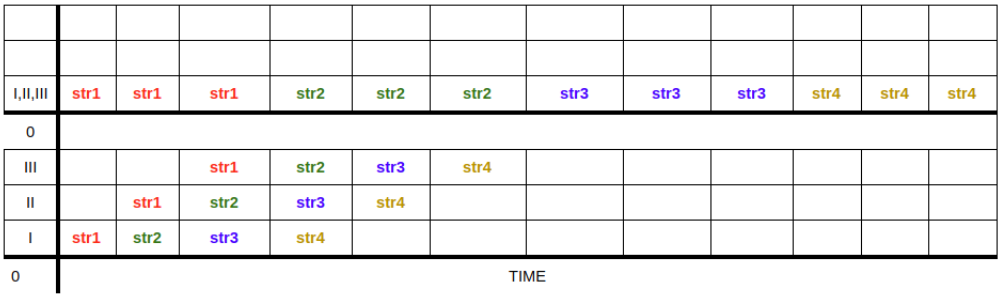
```
{
    First first = new First();
    Second second = new Second();
    Third third = new Third();

    first.SetNext onveyer(second);
    second.SetNext onveyer(third);

    foreach (string word in content)
    {
        first.Enqueue(word);
    }

    List<Thread> threads = new List<Thread>
    {
        new Thread(() => first.Run()),
        new Thread(() => second.Run()),
        new Thread(() => third.Run())
    };
    threads.ForEach(t => t.Start());
    threads.ForEach(t => t.Join());

}
```

Таблица 1. Преобразования строк.

Входная строка	1 этап	2 этап	3 этап
абрикос	абрикоссокирба	абрикос****	аркс**
елка	елкаакле	елка****	ек**

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



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 потребуется
$F(N) = N * 3t$, 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 до 100,000 строк. Также было замерено время на обработку очередей из этих же строк с помощью одного потока, выполняющего три этапа обработки последовательно.

Таблица 2. Результаты замеров времени.

Количество строк в очереди	Время работы в одном потоке (мсек)	Время работы конвейера (мсек)
100	0	0
200	2	0
1269	4	3
1000	5	2
1000	5	2
1500	4	3
1826	9	5
2191	10	6
2629	9	6
3154	14	9
3784	20	10
4540	16	12
5448	16	14
6537	17	18
7844	18	20
9412	21	25
11294	35	35
13552	37	39
16262	33	50
19514	46	66
23416	54	82
28099	63	96
33718	80	121
40461	88	156
48553	101	179
58263	130	232
69915	146	286
83898	180	321
100677	206	370

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

приемущество в три раза, однако экспериментальные замеры показали, что конвейер эффективнее примерно в 1.6 раза. Это происходит из-за того, что на каждом этапе обработка данных занимает неодинаковое время и происходят блокировки, что занимает дополнительное время.

Заключение

В ходе лабораторной работы я реализовал конвейерную обработку данных с помощью потоков на языке программирования C# и сравнил реализованный конвейер с однопоточной обработкой тех же данных.