Государственное образовательное учреждение высшего профессионального

образования

«Московский государственный технический университет

имени Н. Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №6

По теме:

«Исследование конвейерного метода решения задач»

Монахов Д. И.

ИУ7-54

Москва 2018

Оглавление

[1. Аналитическая часть 3](#_Toc532824989)

[1.1. Конвейерный метод 3](#_Toc532824990)

[2. Конструкторская часть 4](#_Toc532824991)

[2.1 Разработка алгоритмов 4](#_Toc532824992)

[3. Технологическая часть 5](#_Toc532824993)

[3.1 Требования к программному обеспечению 5](#_Toc532824994)

[3.2 Средства реализации 5](#_Toc532824995)

[3.3 Листинг кода 6](#_Toc532824996)

[4. Экспериментальная часть 8](#_Toc532824997)

[4.1 Постановка эксперимента 8](#_Toc532824998)

[Заключение 10](#_Toc532824999)

# 1. Аналитическая часть

В этой части рассматривается, что такое конвейерный метод.

## 1.1. Конвейерный метод

При конвейерном методе задача разбивается на последовательность подзадач, каждая из которых зависит от результата решения предыдущей подзадачи. Таким образом для каждой подзадачи выделяется поток, который ее решает. Как только подзадача решена, результат передается в следующий поток, а текущий принимает данные от предыдущего потока. Формируется очередь данных, которая положена на «конвейер». В его начале находятся нерешенные задачи, в середине задачи в процессе решения, а в конце готовые задачи. Видна аналогия с конвейерным производством продукции.

# 2. Конструкторская часть

В этой части будут составлен псевдокод алгоритмов.

## 2.1 Разработка алгоритмов

Псевдокод одного потока

procedure ThreadJob

while(not shouldStop)

subtask = get\_next\_data()

subresult = doJob(subtask)

write\_to\_queue(subresult)

end while

end procedure

# 3. Технологическая часть

В этой части обосновывается выбор языка программирования, необходимых библиотек и будет приведен листинг кодов каждого из алгоритмов.

## 3.1 Требования к программному обеспечению

Для того, чтобы запустить программу, необходимо иметь:

ОС windows 10, объем оперативной памяти не менее 512 мб, среда visual studio 2017.

## 3.2 Средства реализации

В качестве языка программирования был выбран C#. Его выбор основан на следующих причинах:

1. язык программирования C# является частью платформы .NET Framework, которая хорошо документирована, обладает хорошей стабильностью работы, и имеет богатый набор библиотечных решений;
2. C# позволяет программисту уделять больше времени написанию кода и не тратить время на побочные действия;
3. C# просто отлаживать, он не обладает уязвимостями языка C++, например, проблемами с указателями;
4. C# по скорости работы почти не уступает языку C++.

Время выполнения замеряется средством .NET под названием StopWatch;

## 3.3 Листинг кода

Листинг класса Конвейер

class Conveyer : IDisposable

{

public delegate object Job(object previous);

private readonly Queue<object>[] \_dataQueues;

private bool \_stop;

private readonly Queue<object> \_outputObjects = new Queue<object>();

struct ThreadJobArgument

{

public ThreadJobArgument(int index, Job job)

{

JobIndex = index;

Job = job;

}

public int JobIndex;

public Job Job;

}

private void ThreadJob(object argument)

{

ThreadJobArgument arg = (ThreadJobArgument) argument;

int jobIndex = arg.JobIndex;

Job job = arg.Job;

while (!\_stop)

{

object nextData;

lock (\_dataQueues[jobIndex])

{

if(\_dataQueues[jobIndex].Count != 0)

nextData = \_dataQueues[jobIndex].Dequeue();

else

nextData = null;

}

if(nextData == null)

continue;

object result = job(nextData);

lock (\_dataQueues[jobIndex + 1])

{

\_dataQueues[jobIndex + 1].Enqueue(result);

}

}

Debug.WriteLine("Thread " + jobIndex + " stopped");

}

public void StopConveyer()

{

\_stop = true;

}

public Conveyer(Job[] jobs)

{

//var threads = new Thread[jobs.Length];

\_dataQueues = new Queue<object>[jobs.Length + 1];

for (int i = 0; i < jobs.Length; i++)

{

Thread thread = new Thread(ThreadJob);

thread.Start(new ThreadJobArgument(i, jobs[i]));

\_dataQueues[i] = new Queue<object>();

}

\_dataQueues[jobs.Length] = \_outputObjects;

}

public void WriteNext(object nextData)

{

lock (\_dataQueues[0])

{

\_dataQueues[0].Enqueue(nextData);

}

}

public bool HasNextResult()

{

bool flag;

lock (\_outputObjects)

{

flag = \_outputObjects.Count != 0;

}

return flag;

}

public object ReadNextResult()

{

object result;

lock (\_outputObjects)

{

result = \_outputObjects.Dequeue();

}

return result;

}

public void Dispose()

{

StopConveyer();

}

}

# 4. Экспериментальная часть

В этой части лабораторной работы будут поставлены эксперименты над реализованным алгоритмом.

## 4.1 Постановка эксперимента

Пусть имеется задача, которая разбита неравномерно, т.е. время решения каждой подзадачи можно сильно колебаться. Запуск конвейера для такой задачи наглядно покажет, что конвейер устойчив ко времени решения задач и работает эффективнее обычного линейного алгоритма.

Предположим, что задача состоит из трех частей, первая из которых выполнится за 1 секунду, вторая за 3 секунды, а последняя за 5 секунд. При линейном решении очевидно, что суммарное время решения задачи будет равно сумме времени решения всех подзадач. Но в случае использования конвейера простой будет уменьшен. Запустим программу для последовательности из трех чисел. Каждая подзадача будет искусственно приостановлена на 1, 3 и 5 секунд соответственно.

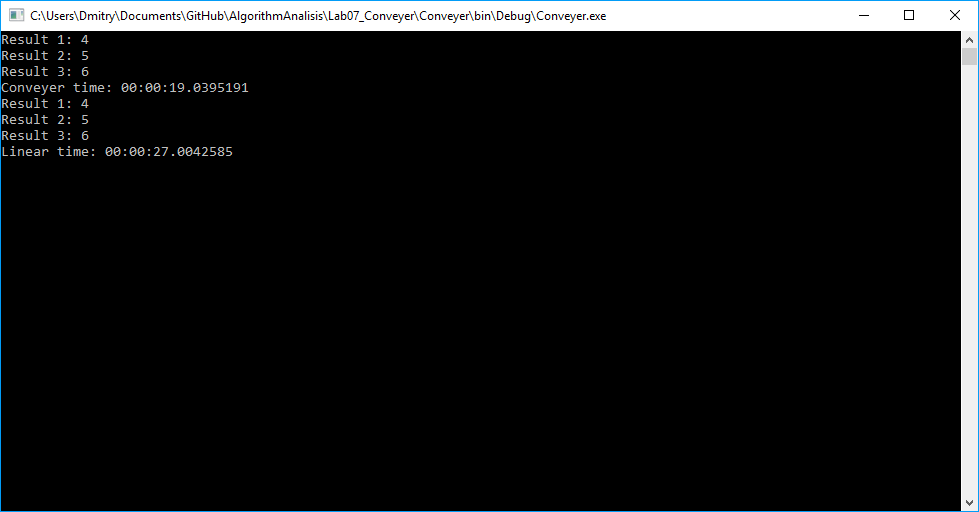


Рисунок 1. Результат для потока из 3х чисел

Покажем, что при увеличении количества входных данных выигрыш по времени также будет увеличиваться.

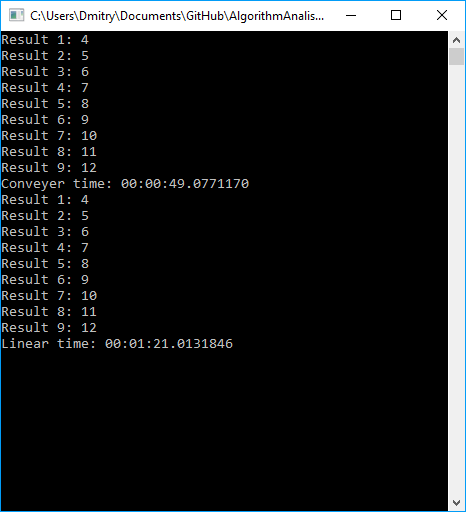


Рисунок 2. Результат для 9 чисел

Если для трех чисел выигрыш был в 30%, то для девяти он составляет уже 40%.

# Заключение

Конвейерный метод эффективен при решении задач на большом непрерывном потоке данных.