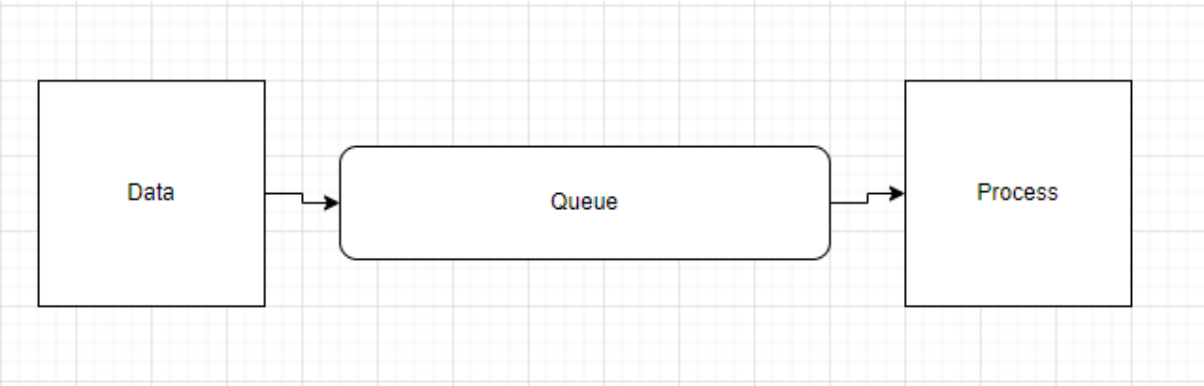
Для повышения производительности приложения используют многопоточность. Потоки позволяют выполнять несколько задач одновременно. Ключевым объектов в c# для работы с потоками является класс Thread, который представляет отдельный поток.

Представим веб-сайт Telega, который позволяет общаться людям путём отправки личных сообщений. Если бы эта соцсеть работала в одном единственном потоке, пользователям пришлось бы ждать пока первый, кто успел занять поток отправит все сообщения.

Смоделируем ситуацию: мы создали сайт для расчета функции Лапласа. У нас есть два пользователя, которые генерируют значения аргумента и отправляют их нам, а мы в свою очередь должны не потерять эти данные и посчитать их все.



Для выполнения такой задачи создадим потоки, которые будут создавать данные (Data). Очередь запросов, чтобы вычислять в первую очередь те данные, которые быстрее пришли. (Queue) И потоки, которые будут вычислять нашу функцию (Process).

1. Создать очередь Queue<double> как статическое поле.
2. Создать метод генерации 10 случайных вещественных чисел в промежутке [0,2), с округлением до 2 знаков после запятой.

```
private static void Generate()
{
    for (int i = 0; i < 10; ++i)
    {
        double value = Math.Round(_random.NextDouble() * 2, 2);
        Console.WriteLine($"{Thread.CurrentThread.Name} : Enqueue {value}");
        _queue.Enqueue(value);
    }
}
```

3. Создать метод получения из очереди значения (нужно дождаться пока в ней появятся данные), и вывода этого значения в консоль.

```
private static void Process()
{
    while (true)
    {
        double value;
        if(_queue.TryDequeue(out value))
        {
            Console.WriteLine($"{Thread.CurrentThread.Name} : Dequeue
{value}");

        }
    }
}
```

4. Создать два потока, которые будут генерировать данные и добавим им имя.

```
Thread thread1 = new Thread(Generate)
{
    Name = "thread1"
};
Thread thread2 = new Thread(Generate)
{
    Name = "thread2"
};
```

5. Создать несколько потоков, которые будут считывать данные из очереди.

```
Thread thread3 = new Thread(Process)
{
    Name = "thread3"
};
Thread thread4 = new Thread(Process)
{
    Name = "thread4"
};
```

6. Запустим все потоки

```
thread1.Start();
thread2.Start();
thread3.Start();
thread4.Start();
```

Мы хотим, чтобы приложение завершилось, когда 20 чисел будут сгенерированы и все значения будут вычислены, то необходимо

7. Дождаться завершения потоков генерации чисел при помощи thread.Join().

```
thread1.Join();
thread2.Join();
```

8. Дождаться, пока очередь не станет пуста.

```
while (_queue.Count != 0) { }
```

Запустим наше приложение и…..

Уникальные значения, записанные в очередь:
0,35 1,87 1,42 1,64 1,34 1,15 1,33 1,51 0,23 0,36 0,47 0,02 1,97 0,5 1,45 1,58 1,71 1,43 1,38 1,31

Уникальные значение, взятые потоками из очереди:
1,87 1,42 1,64 1,34 1,15 1,33 1,51 0,23 0,36 0,47 0,02 1,58 1,43 1,71 1,38 1,31

У нас происходит что-то странное. Некоторые элементы были взяты из очереди по нескольку раз.

Вся проблема в том, что доступ к очереди Queue не синхронизирован (а сам тип Queue не является безопасным в многопоточном окружении) и может происходить следующее:

1. Первый поток читает очередь;
2. Второй поток читает очередь;
3. Первый поток взял из очереди элемент, указатель в очереди на следующий элемент сдвинулся
4. Второй поток, который не прочитал новое значение указателя в очереди также считывает первый элемент и сдвигает указатель на следующий элемент.

Таким образом у нас получается считывать по нескольку раз один и тот же элемент.

Так же может происходить и другой эффект:

1. Два потока считывают последний элемент.
2. Первый поток считал значение, сдвинул указатель и уменьшил количество элементов в очереди на 1, в итоге количество элементов становится равным 0.
3. Второй поток считывает значение, считывает количество элементов, которое для этого потока равно 1, доходит до момента, когда нужно уменьшить значение количества, считывает новое значение, которое равно 0, уменьшает его до -1.
4. Т.к. в очереди проверка на отсутствие элементов выглядит так:

   ```
   if (_size == 0)
   {
       ThrowForEmptyQueue();
   }
   ```

   То дальше потоки будут бесконечно (закольцованно) считывать элементы из очереди.

Чтобы такого не происходило, для многопоточных операций можно использовать коллекции из System.Collections.Concurrent. В нашем случае это ConcurrentQueue<T>.

Также наше приложение не завершается. Потоки можно разделить на два типа: Основные и фоновые. В нашем случае мы создали основные потоки. Эти потоки будут работать до того момента пока они не завершат свою работы. Фоновые потоки принудительно завершаются, когда завершаются все основные потоки в процессе. Т.к. потоки вычисления функции являются основными потоками и находятся в бесконечном цикле, то приложение не завершится само. Для того, чтобы сделать поток фоновым, необходимо установить свойство потока IsBackground в true.

1. Заменим Queue<double> на ConcurrentQueue<double>.
2. Установим свойству потоков IsBackground значение true.

```
Thread thread3 = new Thread(Process)
{
   IsBackground = true,
   Name = "thread3"
};
Thread thread4 = new Thread(Process)
{
   IsBackground = true,
   Name = "thread4"
};
```

3. Убедимся, что сейчас работает всё правильно. Мы не теряем данные пользователей.
4. Потоки можно отправлять в состояние сна на некоторое время. Поэтому в методе генерации чисел на каждой итерации будем отправлять поток в сон на случайное время (для моделирования получения данных).

```
private static void Generate()
{
   for (int i = 0; i < 10; ++i)
   {
       Thread.Sleep(_random.Next(1000));
       double value = Math.Round(_random.NextDouble() * 2, 2);
       Console.WriteLine($"{Thread.CurrentThread.Name} : Enqueue {value}");
       _queue.Enqueue(value);
   }
}
```

5. Добавим вычисление функции и вывод результата в консоль.

```
private static void Process()
{
   while (true)
   {
       if(_queue.TryDequeue(out var value))
       {
```

```
            Console.WriteLine($"{Thread.CurrentThread.Name} :
\u03a6({value:F2}) = {GetIntegral(value)}");
        }
    }
}
```

Чтобы правильно вывелся символ UTF-8, необходимо установить кодировку консоли для вывода на UTF-8.