Практическая работа № 7

1 часть

Создание простого многопоточного приложения, в котором будет реализовано три различных потока

1. Создадим простое консольное приложение и назовем его, к примеру, "SImpleMultithreadingApplication":

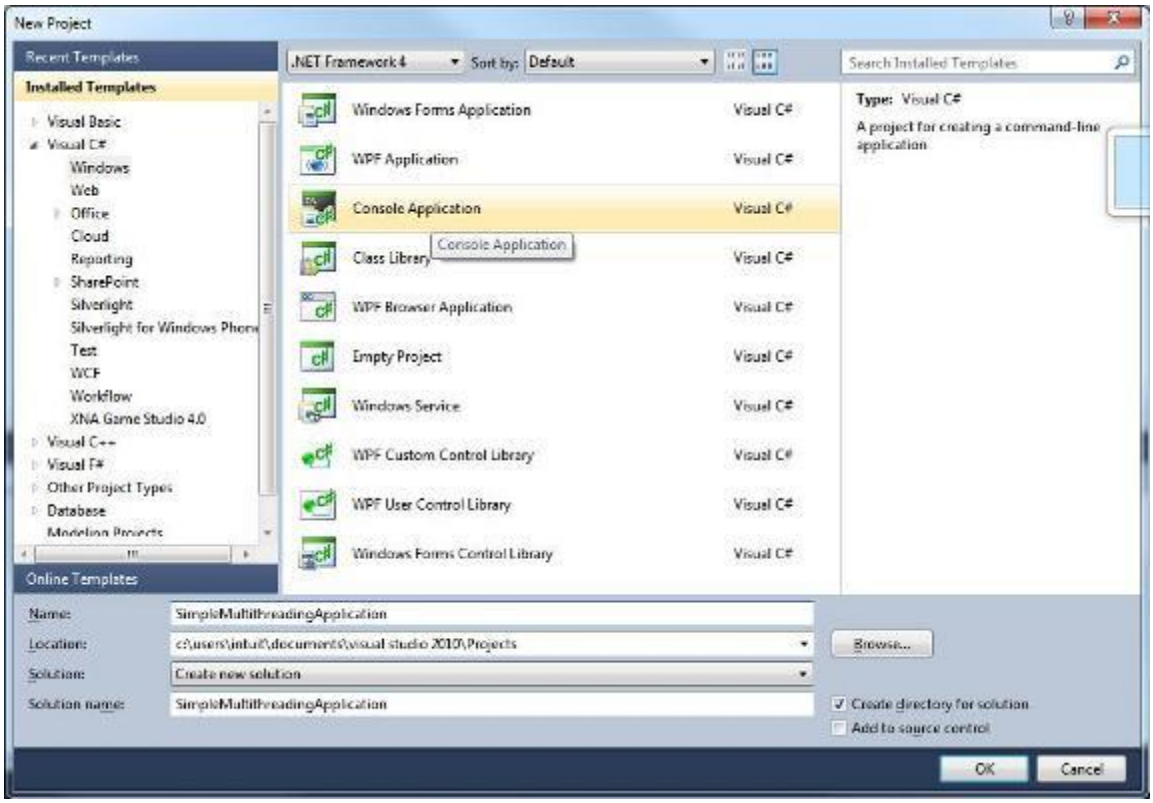

Рис. 1.

2. Далее выведем в однопоточном режиме сообщение "Hello World!" с помощью кода:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace SimpleMultithreadingApplication
{
   class Program
   {
      static void Main(string[] args)
      {
         for (int i = 0; i < 10;i++)
         {
            Console.WriteLine("Hello World!");
         }
  Console.ReadLine();
      }
   }
}
```

Так как, действие выполняется в один поток (метод Main), результат будет следующим:

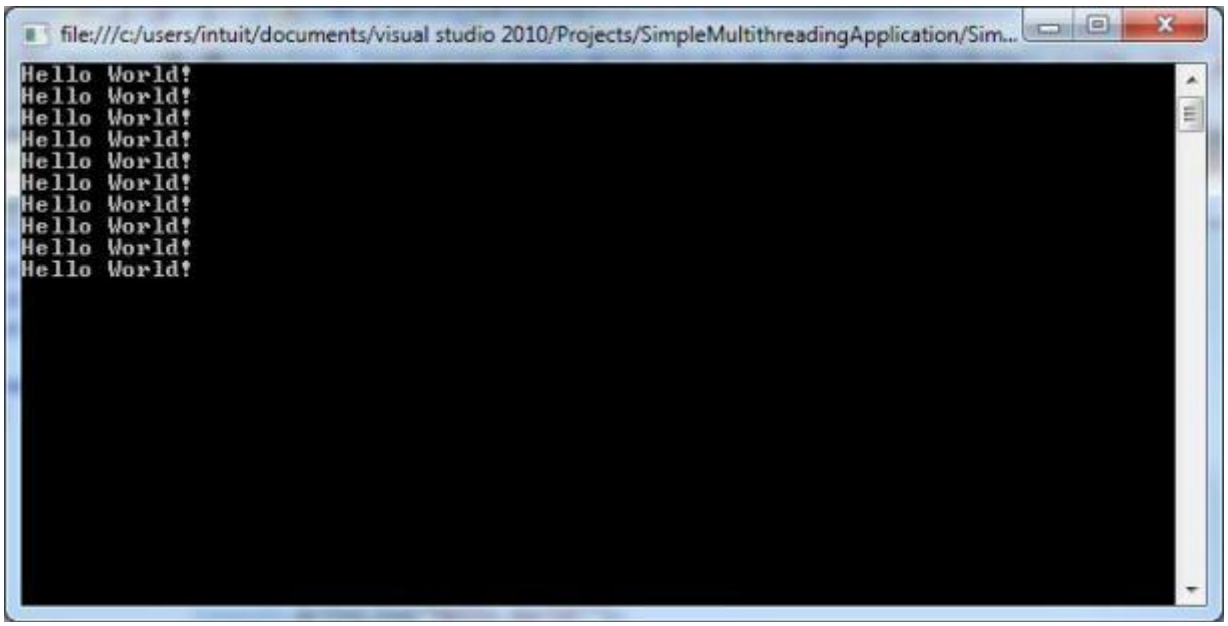

Рис. 2.

3. Для того, что бы использовать многопоточность, используем директиву System.Threading:

```
using System.Threading;
```

4. Теперь, создадим три различных метода, которые будут выполняться в различных потоках:

```
static void FirstThread()
        {
           for (int i=0;i<10;i++)
           {
              Console.WriteLine("Первый поток говорит: Hello!");
           }
        }
        static void SecondThread()
        {
           for (int i = 0; i < 10; i++)
           {
              Console.WriteLine("Второй поток говорит: World!");
           }
        }
        static void ThirdThread()
        {
           for (int i = 0; i < 10; i++)
           {
              Console.WriteLine("Третий поток говорит: Hello World!");
           }
        }
```

5. В методе Main создаем потоки:

```
Thread thread1 = new Thread(new ThreadStart(FirstThread))
Thread thread2 = new Thread(new ThreadStart(SecondThread))
Thread thread3 = new Thread(new ThreadStart(ThirdThread))
```

В нашем случае поток:

- thread1 - вызывает метод FirstThread
- thread2 - вызывает метод SecondThread

- thread3 - вызывает метод ThirdThread

6. Запускаем потоки, с помощью метода Start():

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

7. Добавим на вывод в главном потоке (метод Main), следующие строчки:

```
Console.WriteLine("Главный поток молчит")
Console.WriteLine("Завершение главного потока")
Console.ReadLine()
```

Весь код будет выглядеть следующим образом:

```
static void Main(string[] args)
    {
        Thread thread1 = new Thread(new ThreadStart(FirstThread));
        Thread thread2 = new Thread(new ThreadStart(SecondThread));
        Thread thread3 = new Thread(new ThreadStart(ThirdThread));
        thread1.Start();
        thread2.Start();
        thread3.Start();
        Console.WriteLine("Главный поток молчит");
        Console.WriteLine("Завершение главного потока");
        Console.ReadLine();
     }
```

8. Запустим приложение. Результат будет следующим:

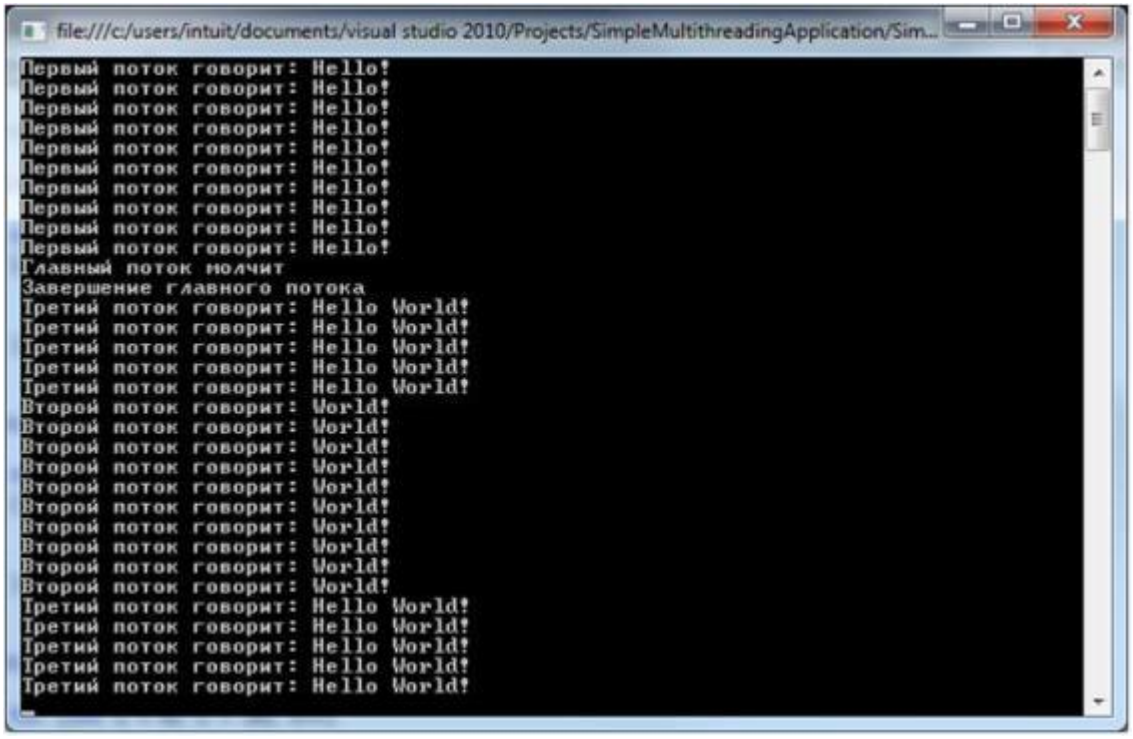

Рис. 3.

Как видно из результата, потоки выполняются с различным интервалом по времени.

9. Для того, что бы задать время блокировки, на выполнение потока, используем метод Sleep(). Для главного потока установим время блокировки 1000мс:

```
Thread.Sleep(1000)
Console.WriteLine("Главный поток молчит")
Console.WriteLine("Завершение главного потока")
```

```
Console.ReadLine()
```

10. Аналогично, установим время блокировки, для дочерних потоков 20мс, 100мс, 90мс:

```
static void FirstThread()
        {
           for (int i=0;i<10;i++)
           {
             Thread.Sleep(20);
               Console.WriteLine("Первый поток говорит: Hello!");
           }
          Console.WriteLine("Завершение первого потока");
        }
        static void SecondThread()
        {
           for (int i = 0; i < 10; i++)
           {
              Thread.Sleep(100);
              Console.WriteLine("Второй поток говорит: World!");
           }
         Console.WriteLine("Завершение второго потока");
        }
        static void ThirdThread()
        {
           for (int i = 0; i < 10; i++)
           {
            Thread.Sleep(90);
              Console.WriteLine("Третий поток говорит: Hello World!");
           }
          Console.WriteLine("Завершение третьего потока");
        }
```

11. Запустим программу. Результат будет следующим:

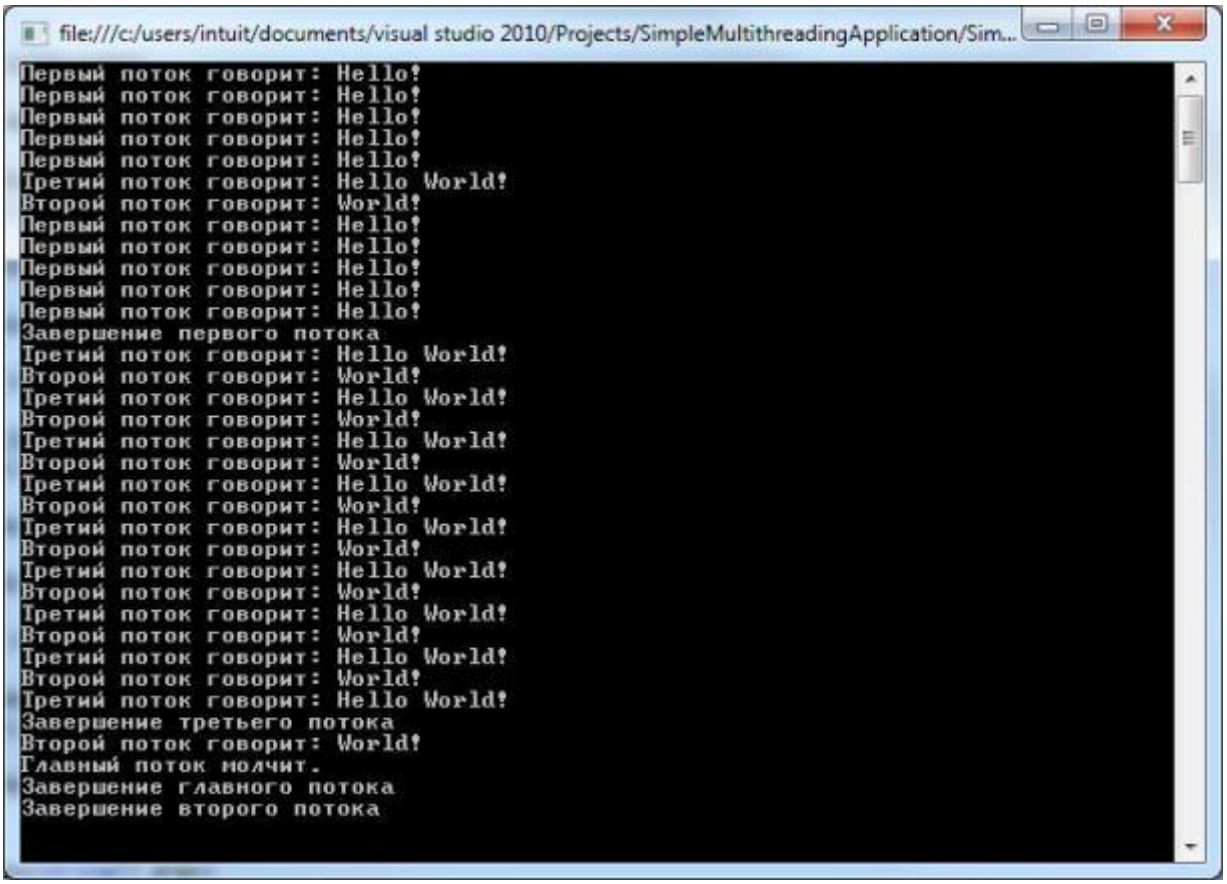

Рис. 4.

Как видно из результата выполнения программы: первый поток завершится первым, т.к время блокировки составляет 20мс. Второй и третий поток будут выполняться почти одновременно, т.к время блокировки составляет 90мс и 100мс. Второй поток и главный поток завершатся почти одновременно т.к общее время блокировки второго и главного потока составляет 1000мс.

Листинг кода программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;
using System.Collections.Concurrent;
namespace ConcurrentCollectionsApplication
{
   class Program
   {
      static BlockingCollection<char> bc;
      static void Producer()
      {
         for (char ch = 'A'; ch <= 'Я'; ch++)
         {
            bc.Add(ch);
            Console.WriteLine("Производится символ " + ch);
         }
      bc.CompleteAdding();
```

```
        }
        static void Consumer()
        {
            char ch;

                while(!bc.IsCompleted)
                {
             if(bc.TryTake(out ch))
                Console.WriteLine("Потребляется символ "+bc.Take());
                }

        }
        static void Main(string[] args)
        {
            bc = new BlockingCollection<char>(4);
            Task Prod = new Task(Producer);
            Task Con = new Task(Consumer);
            Con.Start();
            Prod.Start();
            try
            {
                Task.WaitAll(Con, Prod);
            }
            catch (AggregateException exc)
            {
                Console.WriteLine(exc);
            }
            finally
            {
                Con.Dispose();
                Prod.Dispose();
                bc.Dispose();
            }
            Console.ReadLine();
        }
    }
}
```

Создание простого приложения с использованием многозадачности.

1. Создадим многозадачное консольное приложение и назовем его, к примеру, "SimpleMultitaskingApplication":

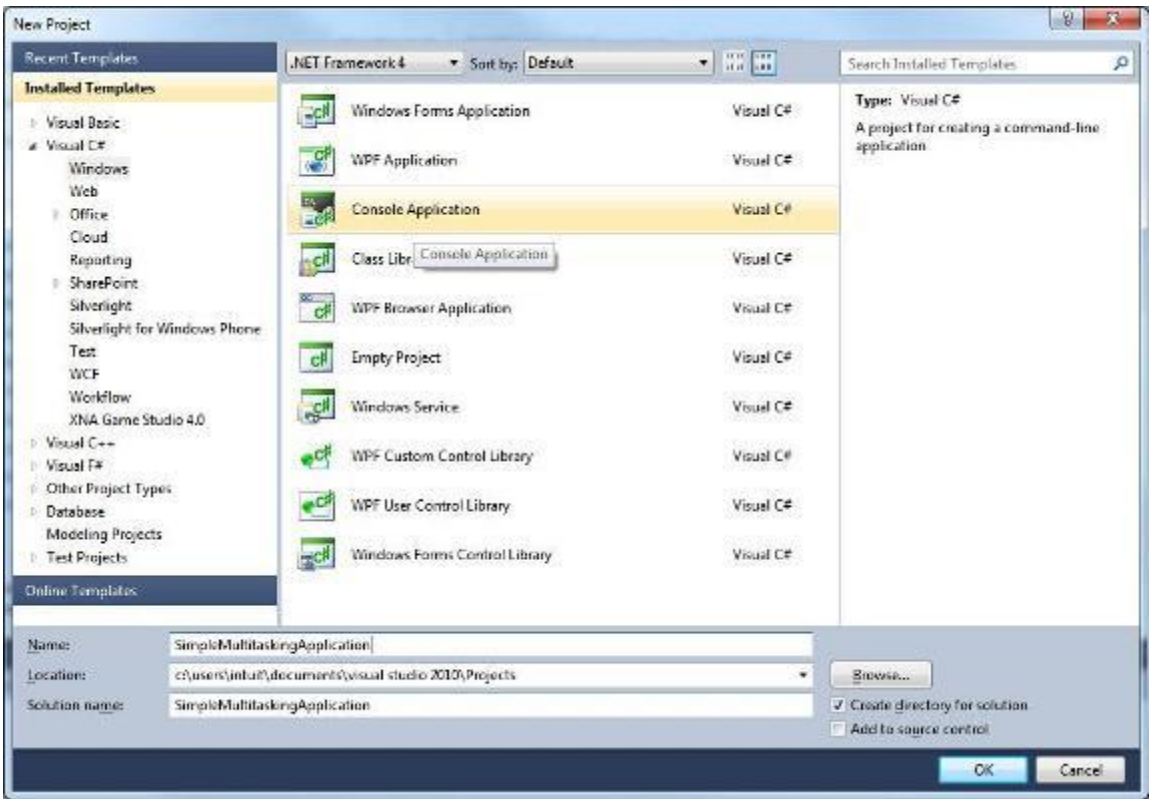

Рис. 5.

2. Создадим статичный метод, который будет выводить на консоль сообщение "Hello world!":

```
private static void printMessage() {
        Console.WriteLine("Hello world!");
    }
```

3. Создадим четыре различных задачи, различными способами:

```
Task task1 = new Task(new Action(printMessage));
      Task task2 = new Task(delegate { Console.WriteLine("Hello world!"); });
       Task task3 = new Task(() => printMessage());
       Task task4 = new Task(() => { Console.WriteLine("Hello world!"); });
```

   - task1 - используем делегат Action и именной метод (в нашем случае printMessage);
   - task2 - используем анонимный делегат;
   - task3 - используем лямбда выражение и именной метод;
   - task4 - используем лямбда выражение и анонимный метод.

4. Запускаем задачи с помощью метода Start():

```
task1.Start()
      task2.Start()
      task3.Start()
      task4.Start()
```

5. Обозначим завершение главного потока (метод Main()) с помощью фрагмента кода:

```
Console.WriteLine("Главный метод завершен.")
      Console.ReadLine()
```

Листинг кода программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;
namespace SimpleMultitaskingApplication
{
   class Program
   {
   private static void printMessage()
 {
      Console.WriteLine("Hello world!");
    }
      static void Main(string[] args)
      {
      Task task1 = new Task(new Action(printMessage));
      Task task2 = new Task(delegate { Console.WriteLine("Hello world!"); });
      Task task3 = new Task(() => printMessage());
      Task task4 = new Task(() => { Console.WriteLine("Hello world!"); });
      task1.Start();
      task2.Start();
      task3.Start();
      task4.Start();
      Console.WriteLine("Главный метод завершен.");
      Console.ReadLine();
    }
     }
   }
```

6. Запустим программу, результат выглядит следующим образом:

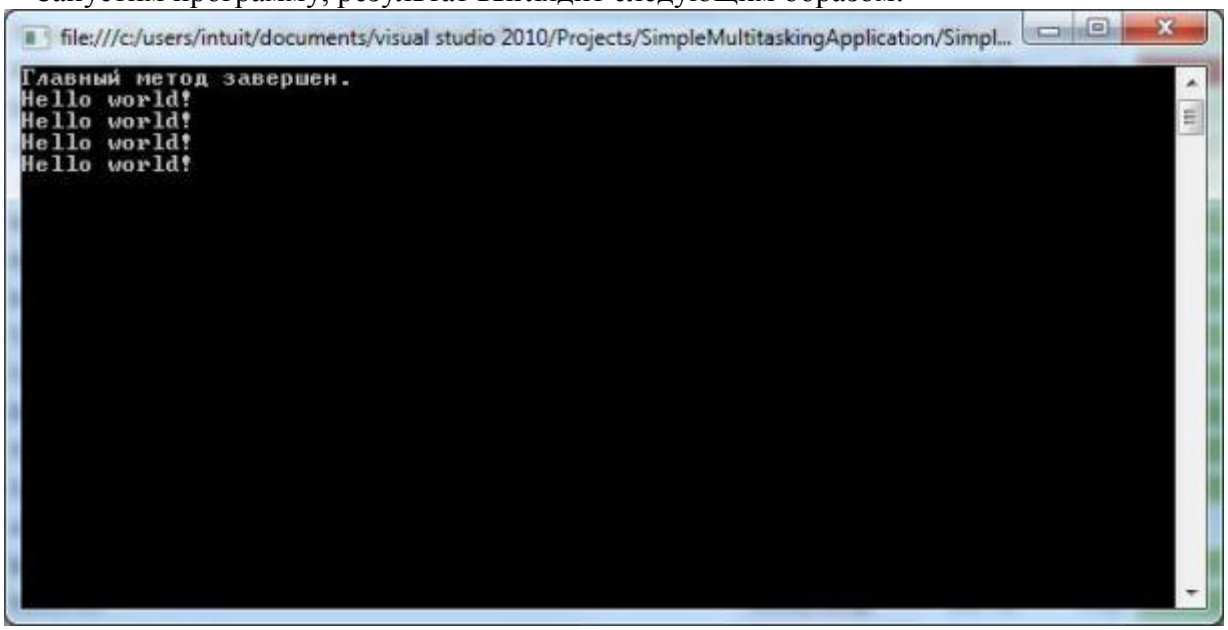

Рис. 6.

7. Модифицируем программу. Изменим метод printMessage():

```
private static string printMessage(string message)
   {
     return message.ToUpper();
   }
```

Данный перегруженный метод теперь возвращает значение типа string, преобразованное в верхний регистр.

8. Создадим задачу, в методе Main, которая бы передавала в метод printMessage значение и возвращала результат выполнения задачи в виде строки:

```
Task <string> message = new Task<string>(mes => printMessage((string) mes), "hello world")
        message.Start()
        Console.WriteLine("Сообщение: " + message.Result)
```

9. Запустим программу. Программа выведет следующий результат на экран:

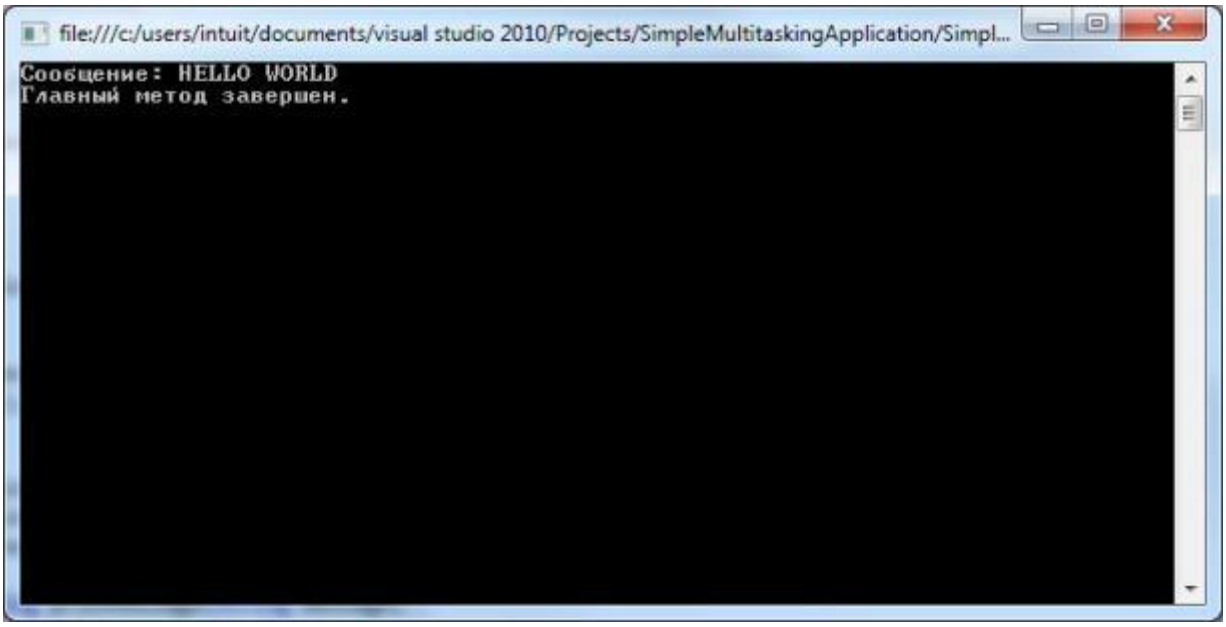

Рис. 7.

Листинг кода программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace SimpleMultitaskingApplication
{
   class Program
   {
      static void Main(string[] args)
      {
         Task<string> message = new Task<string>(mes => printMessage((string)mes), "hello world");
         message.Start();
         Console.WriteLine("Сообщение: " + message.Result);
         Console.WriteLine("Главный метод завершен.");
         Console.ReadLine();
      }
   private static string printMessage(string message)
   {
      return message.ToUpper();
   }
      }
   }
```

10. Теперь, создадим новую задачу, с использованием метода ContinueWith. Данный метод вызывает продолжение выполнения целевой задачи, но уже в другой задаче, в нашем случае задачи message:

```
Task cwt =message.ContinueWith(task =>  Console.WriteLine("Сообщение: " +
message.Result));
```

11. Запустим приложение. Результат будет следующим:

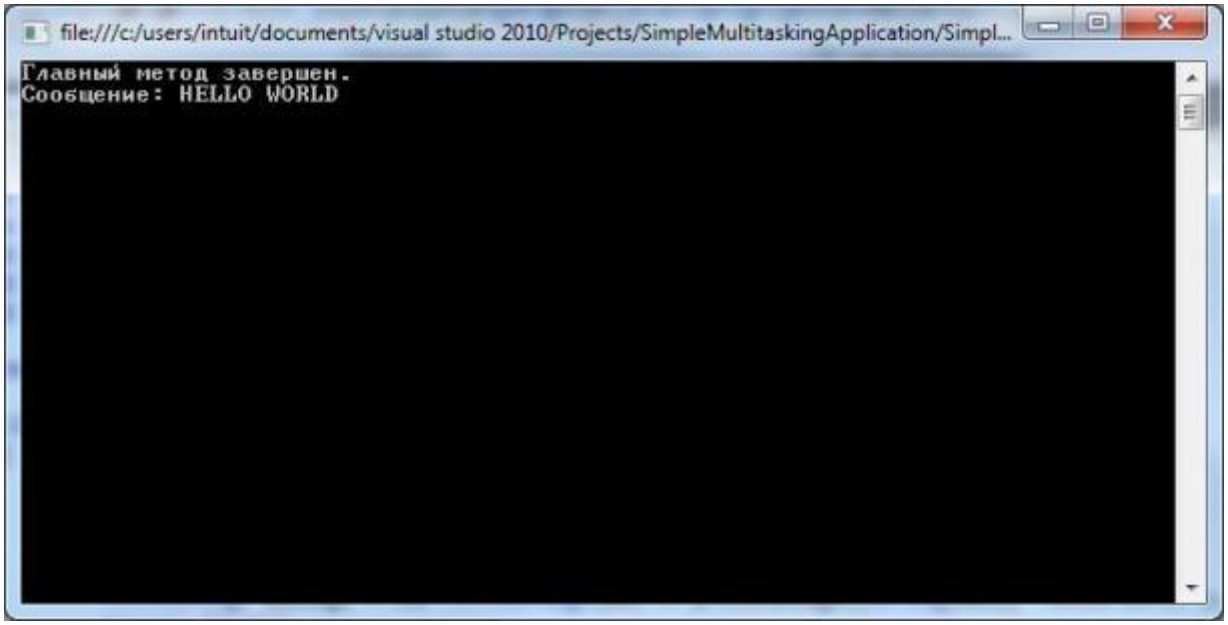

Рис. 8.

Примечание. Как видно из результата выполнения программы, главный метод завершит выполнение раньше, чем дочерни задачи, это связанно с тем, что задача вывода на экран результата (Task cwt) выполнения метода printMessage., выполнится только тогда, когда завершится выполнение задачи Task <string> message.

Листинг кода программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace SimpleMultitaskingApplication
{
   class Program
   {
   private static string printMessage(string message)
   {
      return message.ToUpper();
   }
      static void Main(string[] args)
      {
         Task<string> message = new Task<string>(mes => printMessage((string)mes), "hello
world");
         message.Start();
         Task cwt = message.ContinueWith(task => Console.WriteLine("Сообщение: " +
message.Result));
         Console.WriteLine("Главный метод завершен.");
```

```
            Console.ReadLine();
        }
        }
    }
```

12. Для создания дочерних задач используем следующий фрагмент кода:

```
Task<string[]> message = new Task<string[]>(() =>
   {
       var result = new string[3];
       new Task(() => result[0] = printMessage("Hello"),
TaskCreationOptions.AttachedToParent).Start();
      new Task(() => result[1] = printMessage("World"),
TaskCreationOptions.AttachedToParent).Start();
       new Task(() => result[2] = printMessage("Hello world!"),
TaskCreationOptions.AttachedToParent).Start();
      return result;
    });
```

Данный фрагмент кода возвращает строковый массив, который формируется с помощью дочерних задач.

13. Теперь необходимо вывести строковый массив на экран с помощью фрагмента кода:

```
var cwt =message.ContinueWith(mes =>  Array.ForEach(mes.Result,Console.WriteLine))
       message.Start()
```

14. Запустим программу. В результате на экране отобразится следующее:

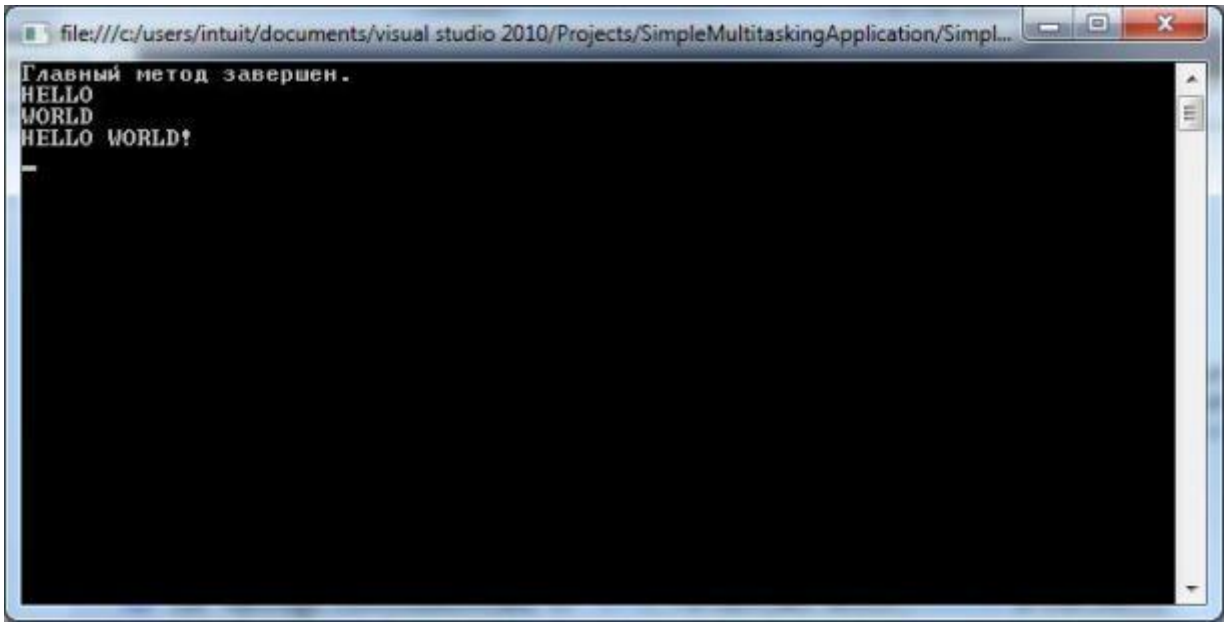

Рис. 9.

Листинг кода программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace SimpleMultitaskingApplication
{
   class Program
   {
```

```
        static void Main(string[] args)
        {
            Task<string[]> message = new Task<string[]>(() =>
                {
                    var result = new string[3];
                    new Task(() => result[0] = printMessage("Hello"),
TaskCreationOptions.AttachedToParent).Start();
                    new Task(() => result[1] = printMessage("World"),
TaskCreationOptions.AttachedToParent).Start();
                    new Task(() => result[2] = printMessage("Hello world!"),
TaskCreationOptions.AttachedToParent).Start();
                    return result;
                });
            var cwt =message.ContinueWith(mes =>
Array.ForEach(mes.Result,Console.WriteLine));
            message.Start();
        Console.WriteLine("Главный метод завершен.");
        Console.ReadLine();
        }
    private static string printMessage(string message)
    {
        return message.ToUpper();
    }
        }
    }
```