Quick answers to common problems

Multithreading in C# 5.0 Cookbook

Over 70 recipes to help you learn asynchronous and parallel programming with C# 5.0 quickly and efficiently

Eugene Agafonov

[PACKT] PUBLISHING

Multithreading in C# 5.0 Cookbook

Over 70 recipes to help you learn asynchronous and parallel programming with C# 5.0 quickly and efficiently

Eugene Agafonov

[PACKT] PUBLISHING

BIRMINGHAM - MUMBAI

Multithreading in C# 5.0 Cookbook

First published: November 2013

Production Reference: 1191113

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-84969-764-4

www.packtpub.com

Cover Image by Aniket Sawant (aniket_sawant_photography@hotmail.com)

Credits

Author

Eugene Agafonov

Reviewers

Mickael Ferrer

Chad McCallum

Philip Pierce

Acquisition Editor

James Jones

Lead Technical Editor

Chalini Snega Victor

Technical Editors

Menza Mathew

Pooja Nair

Copy Editors

Brandt D'Mello

Mradula Hegde

Gladson Monteiro

Sayanee Mukherjee

Aditya Nair

Karuna Narayanan

Kirti Pai

Laxmi Subramanian

Project Coordinator

Apeksha Chitnis

Proofreader

Clyde Jenkins

Indexer

Marriammal Chettiyar

Production Coordinator

Pooja Chiplunkar

Cover Work

Pooja Chiplunkar

About the Author

Eugene Agafonov leads a web development department at ABBYY, and lives and works in Moscow. He has over 15 years of professional experience in software development and started to work with C# from the time it was in its beta version. He is a Microsoft MVP in ASP.NET since 2006, and he often speaks at local software development conferences, such as TechEd Russia, about cutting-edge technologies in the Modern Web and server-side application development. His main professional interests are cloud-based software architecture, scalability, and reliability. Eugene is a huge fan of football and plays the guitar with a local rock band. You can reach him at his personal blog `eugeneagafonov.com` or with his twitter handle `@eugene_agafonov`.

About the Reviewers

Mickael Ferrer is a geek who has played with a lot of technologies through the years; He is the jack of all trades, but master of none.He specialized in .Net and C# development, in particular, for extending Excel.He spent much of his short professional career in the financial industry as a front-office developer.He recently started a self-employed training business for .Net developers.He randomly writes stuff on his technical blog at `pragmateek.com`

Chad McCallum is a Saskatchewan computer geek and an ASP.NET MVP with over seven years of .NET experience. After graduating from the Computer Systems Technology course at SIAST in Saskatoon, he picked up contracting until he could pester iQmetrix to give him a job, where he's been for the last seven years. He had a brief stint in Vancouver, working on interactive retail software. Since then, he's come back to Regina, SK, where he's started HackREGINA, a hackathon organization aimed at strengthening the developer community while coding and drinking beer. Somehow, between his real-life job and sleep, he managed to publish a Pluralsight course on 10 Ways to Build Web Services in .NET. His current focus is on single-page applications with JavaScript. Between random app ideas, he tries to learn a new technology every week; you can see the results on `www.rtigger.com`.

Philip Pierce is a software developer with twenty years of experience in mobile, web, desktop, and server development, database design and management, and game development. His background includes creating A.I. for games and business software, converting AAA games among various platforms, developing multithreaded applications, and creating patented client/server communication technologies.

Philip has won several hackathons, including Best Mobile App at the AT&T Developer Summit 2013, and a runner up for Best Windows 8 App at PayPal's Battlethon Miami. His most recent project was converting Rail Rush and Temple Run 2 from the Android platform to Arcade platforms.

Philip's portfolios can be found at `http://www.rocketgamesmobile.com` and `http://www.philippiercedeveloper.com`.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

To my dearly beloved wife Helen and son Nikita

Table of Contents

Preface

Not so long ago, a typical personal computer CPU had only one computing core, and the power consumption was enough to cook fried eggs on it. In 2005, Intel introduced its first multiple-core CPU, and since then computers started developing in a different direction. Low power consumption and a number of computing cores became more important than a row computing core performance. This lead to programming paradigm changes as well. Now we need to learn how to use all CPU cores effectively to achieve the best performance, and at the same time, save battery power by running only the programs we need at a particular time. Besides that, we need to program server applications in a way to use multiple CPU cores or even multiple computers as efficiently as possible to support as many users as we can.

To be able to create such applications, you have to learn to use multiple CPU cores in your programs effectively. If you use the Microsoft .NET development platform and C# programming language, this book will be a perfect starting point for programming applications that have good performance and responsiveness.

The purpose of this book is to provide you with a step-by-step guide for multithreading and parallel programming in C#. We will start with the basic concepts, going through more and more advanced topic based on the information from previous chapters, and end with real-world parallel programming patterns and Windows Store application samples.

What this book covers

Chapter 1, *Threading Basics*, introduces basic operations with threads in C#. It explains what a thread is, the pros and cons of using threads, and other important thread aspects.

Chapter 2, *Thread Synchronization*, describes thread interaction details. You will learn why we need to coordinate threads together and the different ways of organizing thread coordination.

Chapter 3, *Using a Thread Pool*, explains a thread pool concept. It shows how to use a thread pool, how to work with asynchronous operations, and good and bad practices of using a thread pool.

Chapter 4, *Using Task Parallel Library*, is a deep dive into a Task Parallel Library framework. This chapter outlines every important aspect of TPL, including tasks combination, exceptions management, and operations cancellation.

Chapter 5, *Using C# 5.0*, explains in detail the new C# 5.0 feature – asynchronous methods. You will find out what async and await keywords mean, how to use them in different scenarios, and how await works under the hood.

Chapter 6, *Using Concurrent Collections*, describes standard data structures for parallel algorithms included in the .NET Framework. It goes through sample programming scenarios for each data structure.

Chapter 7, *Using PLINQ*, is a deep dive into the Parallel LINQ infrastructure. The chapter describes task and data parallelism, parallelizing a LINQ query, tweaking parallelism options, partitioning a query, and aggregating parallel query result.

Chapter 8, *Reactive Extensions*, explains how and when to use the Reactive Extensions framework. You will learn how to compose events and how to perform a LINQ query against an event sequence.

Chapter 9, *Using Asynchronous I/O*, covers in detail the asynchronous I/O process including files, networks, and database scenarios.

Chapter 10, *Parallel Programming Patterns*, outlines the common parallel programming problem solutions.

Chapter 11, *There's More*, covers the aspects of programming asynchronous applications for Windows 8. You will learn how to work with Windows 8 asynchronous APIs, and how to perform background work in Windows Store applications.

What you need for this book

For most of the recipes, you will need Microsoft Visual Studio Express 2012 for Windows Desktop. Recipes in Chapter 11 will require Windows 8 and Microsoft Visual Studio Express 2012 for Windows 8 to compile Windows Store applications.

Who this book is for

Multithreading in C# 5.0 Cookbook is written for existing C# developers with little or no background in multithreading, and asynchronous and parallel programming. The book covers these topics from basic concepts to complicated programming patterns and algorithms using C# and .NET ecosystem.

Conventions

In this book, you will find a number of styles of text that distinguish among different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: " When we construct a thread, an instance of the `ThreadStart` or `ParameterizedThreadStart` delegate is passed to the constructor."

A block of code is set as follows:

```
static void PrintNumbers()
{
  Console.WriteLine("Starting...");
  for (int i = 1; i < 10; i++)
  {
    Console.WriteLine(i);
  }
}
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Start Visual Studio 2012. Create a new C# **Console Application** project."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic in which you have expertise, and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Threading Basics

In this chapter, we will cover the basic tasks for working with threads in C#.
You will learn about:

- Creating a thread in C#
- Pausing a thread
- Making a thread wait
- Aborting a thread
- Determining thread state
- Thread priority
- Foreground and background threads
- Passing parameters to a thread
- Locking with a C# lock keyword
- Locking with a Monitor construct
- Handling exceptions

Introduction

At some point of time in the past, the common computer had only one computing unit and could not execute several computing tasks simultaneously. However, operating systems could already work with multiple programs simultaneously, implementing the concept of multitasking. To prevent the possibility of one program taking control of the CPU, forever causing other applications and the operating system itself to hang, the operating systems had to split a physical computing unit across a few virtualized processors in some way and give a certain amount of computing power to each executing program. Moreover, an operating system must always have priority access to the CPU and should be able to prioritize CPU access to different programs. A thread is an implementation of this concept. It could be considered a virtual processor given to the one specific program that runs it independently.

Remember that a thread consumes a significant amount of operating system resources. Trying to share one physical processor across many threads will lead to a situation where an operating system is busy just managing threads instead of running programs.

Therefore, while it was possible to enhance computer processors, making them execute more and more commands per second, working with threads was usually an operating system task. There was no sense in trying to compute some tasks in parallel on a single-core CPU because it would take more time than running those computations sequentially. However, when processors started to have more computing cores, older programs could not take advantage of this because they just used one processor core.

To use a modern processor's computing power effectively, it is very important to be able to compose a program in a way that it can use more than one computing core, which leads to organizing it as several threads communicating and synchronizing with each other.

The recipes in this chapter will focus on performing some very basic operations with threads in the C# language. We will cover a thread's lifecycle, which includes creating, suspending, making a thread wait, and aborting a thread, and then we will go through basic synchronization techniques.

Creating a thread in C#

Throughout the following recipes, we will use Visual Studio 2012 as the main tool to write multithreaded programs in C#. This recipe will show you how to create a new C# program and use threads in it.

There are free Visual Studio 2012 Express editions, which can be downloaded from the Microsoft website. We will need Visual Studio 2012 Express for Windows Desktop for most of the examples and Visual Studio 2012 Express for Windows 8 for Windows 8-specific recipes.

Getting ready

To work through this recipe, you will need Visual Studio 2012. There are no other prerequisites. The source code for this recipe can be found at `BookSamples\Chapter1\Recipe1`.

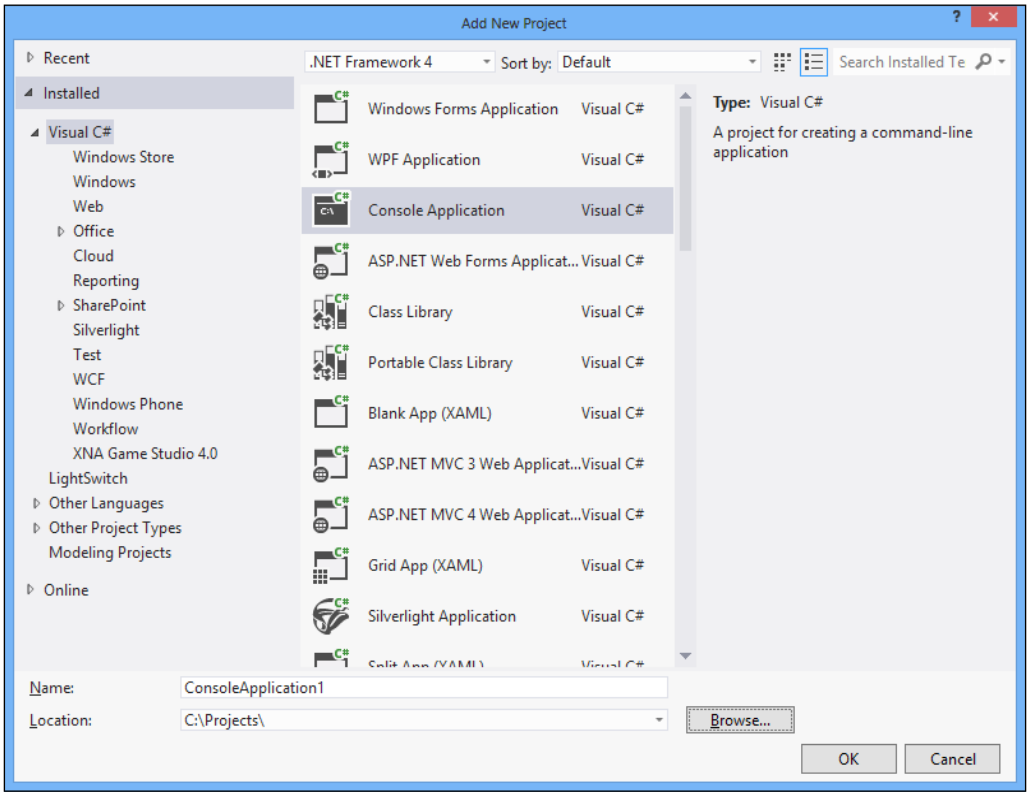
Downloading the example code

You can download the example code files for all Packt books you have purchased through your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

How to do it...

To understand how to create a new C# program and use threads in it, perform the following steps:

1. Start Visual Studio 2012. Create a new C# **Console Application** project.
2. Make sure that the project uses .NET Framework 4.0 or higher version.



3. In the `Program.cs` file add the following `using` directives:

```
using System;
using System.Threading;
```

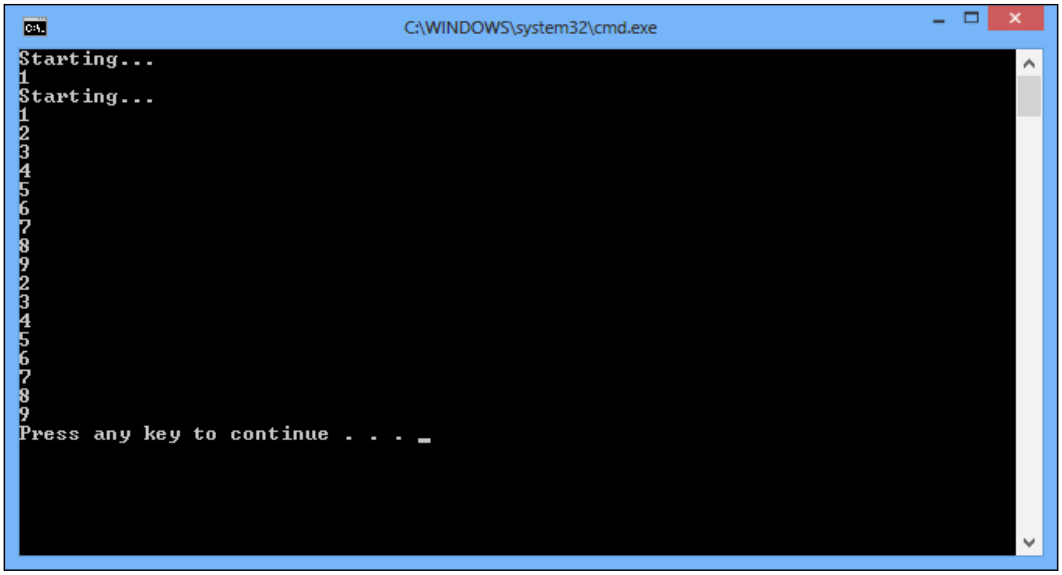
4. Add the following code snippet below the `Main` method:

```
static void PrintNumbers()
{
  Console.WriteLine("Starting...");
  for (int i = 1; i < 10; i++)
  {
    Console.WriteLine(i);
  }
}
```

5. Add the following code snippet inside the `Main` method:

```
Thread t = new Thread(PrintNumbers);
t.Start();
PrintNumbers();
```

6. Run the program. The output will be something like:



How it works...

In steps 1 and 2 we created a simple console application in C# using .Net Framework version 4.0. Then in step 3 we included the namespace `System.Threading`, which contains all the types needed for the program.

An instance of a program is being executed can be referred to as a process. A process consists of one or more threads. This means that when we run a program, we always have one main thread that executes the program code.

In step 4 we defined the method `PrintNumbers`, which will be used in both the main and newly created threads. Then in step 5, we created a thread that runs `PrintNumbers`. When we construct a thread, an instance of the `ThreadStart` or `ParameterizedThreadStart` delegate is passed to the constructor. The C# compiler is creating this object behind the scenes when we just type the name of the method we want to run in a different thread. Then we start a thread and run `PrintNumbers` in the usual manner on the main thread.

As a result, there will be two ranges of numbers from 1 to 10 randomly crossing each other. This illustrates that the `PrintNumbers` method runs simultaneously on the main thread and on the other thread.

Pausing a thread

This recipe will show you how to make a thread wait for some time without wasting operating system resources.

Getting ready

To work through this recipe, you will need Visual Studio 2012. There are no other prerequisites. The source code for this recipe can be found at `BookSamples\Chapter1\Recipe2`.

How to do it...

To understand how to make a thread wait without wasting operating system resource, perform the following steps:

1. Start Visual Studio 2012. Create a new C# **Console Application** project.
2. In the `Program.cs` file add the following `using` directives:

```
using System;
using System.Threading;
```

3. Add the following code snippet below the `Main` method:

```
static void PrintNumbers()
{
  Console.WriteLine("Starting...");
  for (int i = 1; i < 10; i++)
  {
    Console.WriteLine(i);
  }
}
static void PrintNumbersWithDelay()
{
  Console.WriteLine("Starting...");
  for (int i = 1; i < 10; i++)
  {
    Thread.Sleep(TimeSpan.FromSeconds(2));
    Console.WriteLine(i);
  }
}
```

4. Add the following code snippet inside the `Main` method:

```
Thread t = new Thread(PrintNumbersWithDelay);
t.Start();
PrintNumbers();
```

5. Run the program.

How it works...

When the program is run, it creates a thread that will execute a code in the `PrintNumbersWithDelay` method. Immediately after that, it runs the `PrintNumbers` method. The key feature here is adding the `Thread.Sleep` method call to a `PrintNumbersWithDelay` method. It causes a thread executing this code to wait a specified amount of time (two seconds in our case) before printing each number. While a thread is sleeping, it uses as little CPU time as possible. As a result, we will see that the code in the `PrintNumbers` method that usually runs later will be executed before the code in the `PrintNumbersWithDelay` method in a separate thread.

Making a thread wait

This recipe will show you how a program can wait for some computation in another thread to complete to use its result later in the code. It is not enough to use `Thread.Sleep` because we don't know the exact time the computation will take.

Getting ready

To work through this recipe, you will need Visual Studio 2012. There are no other prerequisites. The source code for this recipe can be found at `BookSamples\Chapter1\Recipe3`.

How to do it...

To understand how a program can wait for some computation in another thread to complete to use its result later, perform the following steps:

1. Start Visual Studio 2012. Create a new C# **Console Application** project.
2. In the `Program.cs` file, add the following `using` directives:

```
using System;
using System.Threading;
```

3. Add the following code snippet below the `Main` method:

```
static void PrintNumbersWithDelay()
{
  Console.WriteLine("Starting...");
  for (int i = 1; i < 10; i++)
  {
    Thread.Sleep(TimeSpan.FromSeconds(2));
    Console.WriteLine(i);
  }
}
```

4. Add the following code snippet inside the `Main` method:

```
Console.WriteLine("Starting...");
Thread t = new Thread(PrintNumbersWithDelay);
t.Start();
t.Join();
Console.WriteLine("Thread completed");
```

5. Run the program.

How it works...

When the program is run, it runs a long-running thread that prints out numbers and waits two seconds before printing each number. But in the main program, we called the `t.Join` method, which allows us to wait for thread `t` to complete. When it is complete, the main program continues to run. With the help of this technique, it is possible to synchronize execution steps between two threads. The first one waits until another one is complete and then continues to work. While the first thread is waiting, it is in a blocked state (as it is in the previous recipe when you call `Thread.Sleep`).

Aborting a thread

In this recipe, we will describe how to abort another thread's execution.

Getting ready

To work through this recipe, you will need Visual Studio 2012. There are no other prerequisites. The source code for this recipe can be found at `BookSamples\Chapter1\Recipe4`.

How to do it...

To understand how to abort another thread's execution, perform the following steps:

1. Start Visual Studio 2012. Create a new C# **Console Application** project.
2. In the `Program.cs` file, add the following `using` directives:

```
using System;
using System.Threading;
```

3. Add the following code snippet below the `Main` method:

```
static void PrintNumbersWithDelay()
{
  Console.WriteLine("Starting...");
  for (int i = 1; i < 10; i++)
  {
    Thread.Sleep(TimeSpan.FromSeconds(2));
    Console.WriteLine(i);
  }
}
```