WORKING *with* RUBY THREADS

Jesse Storimer

Copyright (C) 2013 Jesse Storimer.

This book is dedicated to Sara, Inara, and Ora, who make it all worthwhile.

Chapter 0

Introduction

My story

When I joined the Ruby community, I had little understanding of multi-threaded programming.

At that time, almost everyone was using MRI, the original implementation of Ruby. Its threading implementation used green threads. I didn't know what this meant, but people didn't seem to take it too seriously, so I didn't either. At that time, I didn't even know where to begin asking questions when it came to Ruby concurrency.

I had heard the term 'thread safety,' and I knew it was a generally bad thing, but that was the extent of my knowledge.

I still remember the moment that I actually became afraid of thread safety. It was when I read an article[1] which pointed out that the `||=` operator in Ruby is not thread-safe. Even though the application I worked on wasn't running in multiple threads, I was now paranoid about using this operator.

In spite of my trepidation, I just couldn't believe that `||=` was inherently unsafe. It was everywhere in our codebase! It was this trepidation, and the realization that there are

1. http://coderrr.wordpress.com/2009/04/29/is-not-thread-safe-neither-is-hashnew-hk/

fundamentals I knew nothing about, that led me to learn about processes, threads, networking, and all that fun stuff.

In this book, I'll help you through the same trepidation. I'll show you how, in certain contexts, the `||=` operator *can* be not thread-safe, but you certainly don't need to stop using it.

Why care?

If you're reading this book on any kind of modern computing device, it's likely that it has more than one CPU core. And chances are the device you buy next year will have more cores than the one you have today.

Unfortunately, just adding more CPU cores doesn't necessarily make all your code run faster. **Your code must be architected to take advantage of multiple CPU cores** using some concurrency mechanism. If you're not doing this, you might as well be running on ten-year-old hardware.

Operating system processes have been the de facto concurrency mechanism for decades. With processes, if you want more concurrency, then start more processes! This has been the norm in the Ruby community for many years.

So, why threads?

The promise of multi-threading

The promise of multi-threading has always been cheaper concurrency. Cheaper than processes, that is.

Spawning a thread incurs much less overhead than spawning a process.

The primary difference between using processes versus threads is the way that memory is handled. At a high level, processes copy memory, while threads share memory. This makes process spawning slower than thread spawning, and leads to processes consuming more resources once running. Overall, threads incur less overhead than processes.

This smaller overhead means **threads can you give more 'units' of concurrency for the available resources.** Of course, this comes with a cost: introducing multiple threads requires that your code be thread-safe.

Despite the Ruby community's love for process-based solutions, there is a shift happening. The default MRI implementation has always had, even to this day, a threading implementation that limits parallel execution. This stifled community support for a long time.

But more and more people are becoming aware of the promise of multi-threading. More opportunities, and more choices, are becoming available in the community, opportunities like higher throughput and more concurrency, while using less resources. Now's the time to educate yourself and take advantage of these opportunities.

What to expect

In this book I'm providing you with lots of small code samples intended to illustrate key concepts. Please run them in your own console and play with them. Taking the examples and then tweaking them to test your hypotheses is a fantastic way to learn. Look in the included `code/` folder for all of the snippets to avoid copy/paste issues.

The first part of the book focuses on basic concurrency-related topics. Where possible, I attempt to generalize, so that what you learn here can also be applied when you're programming with other languages or just pondering code in general.

That being said, the primary focus of the book is multi-threaded programming *in Ruby*.

The second part of the book dives deeper into what the Ruby language offers to support multi-threaded programming.

The book ends with a hands-on tutorial, where you walk through implementing a concurrent program using several approaches.

There are no clear dividing lines between these three parts, and there's certainly some overlap. **The goal of the book is to give you the necessary knowledge so that you can make good decisions about concurrency for your application.** This includes making you comfortable with the idea of multi-threaded programming, dispelling myths that may be floating around the community, and showing you what tools Ruby offers to aid you.

Which version of Ruby is used?

This book studies three different Ruby language implementations.

1. MRI, version 2.0.0 (all code is also 1.9.3 compatible).
2. JRuby, version 1.7.4.
3. Rubinius, version 2.0.0-rc1.

For JRuby and Rubinius, it's assumed that you're running them in 1.9 language mode. When relevant, I'll show sample code output from all three implementations.

The reason to include multiple Ruby implementations is that they have very different stories when it comes to multi-threading, which can lead to differences in behaviour.