1 Introduction

Written by Scott Grosch

Performance. Responsiveness. They're not sexy tasks. When done properly, nobody is going to thank you. When done incorrectly, app retention is going to suffer and you'll be dinged during your next yearly performance review.

There are a multitude of ways in which an app can be optimized for speed, performance and overall responsiveness. This book will focus on the topic of **concurrency**.

What is concurrency?

Wikipedia defines concurrency as "the decomposability property of a program, algorithm, or problem into order-independent or partially-ordered components or units." What this means is looking at the logic of your app to determine which pieces can run at the same time, and possibly in a random order, yet still result in a correct implementation of your data flow.

Moderns devices almost always have more than a single CPU, and Apple's iPhones have been dual core since 2011. Having more than one core means they are capable of running more than a single task at the same time. By splitting your app into logical "chunks" of code you enable the iOS device to run multiple parts of your program at the same time, thus improving overall performance.

Why use concurrency?

It's critical to ensure that your app runs as smoothly as possible and that the end user is not ever forced to wait for something to happen. A second is a minuscule amount of time for most everything not related to a computer. However, if a human has to wait a second to see a response after taking an action on a device like an iPhone, it feels like an eternity. "It's too slow" is one of the main contributors to your app being uninstalled.

Scrolling through a table of images is one of the more common situations wherein the end user will be impacted by the lack of concurrency. If you need to download an image from the network, or perform some type of image processing before displaying it, the scrolling will stutter and you'll be forced to display multiple "busy" indicators instead of the expected image.

A beneficial side effect to using concurrency is that it helps you to spend a bit more time thinking about your app's overall architecture. Instead of just writing massive methods to "get the job done" you'll find yourself naturally writing smaller, more manageable methods that can run concurrently.

How to use concurrency

That's the focus of this book! At a high level you need to structure your app so that some tasks can run at the same time. Multiple tasks that modify the same resource (i.e., variable) can't run at the same time, unless you make them thread safe.

Tasks which access different resources, or *read-only* shared resources, can all be accessed via different threads to allow for much faster processing.

This book will focus on the two main ways that iOS provides you with the ability to run code concurrently. The first section on **Grand Central Dispatch** will cover the common scenarios where you will find yourself being able to implement concurrency. You'll learn how to run tasks in the background, how to group tasks together and how to handle restricting the number of tasks that can run at once. By the end of the first section you'll also have a strong grasp of the dangers of concurrency and how to avoid them.

In the second section you'll focus on the `Operation` class. Built on top of Grand Central Dispatch, operations allow for the handling of more complex scenarios such as reusable code to be run on a background thread, having one thread depend on another, and even canceling an operation before it's started or completed.

Most modern programming languages provide for some form of concurrency and Swift is of course no exception. Different languages use widely different mechanisms for handling concurrency. C# and Typescript, for example use an *async/await* pattern, whereas Swift uses closures to handle what runs on another thread. Swift 5 originally had plans to implement the more common *async/await* pattern but it was removed from the specification until the next release.

Where to go from here?

Well to the next page of course! Hopefully as you work through the following chapters you'll gain an appreciation for what concurrency can do for your app and why your end users will appreciate the extra effort you put into making the app perform as fast as possible. Knowing when to use Grand Central Dispatch as opposed to an `Operation` subclass early in the app lifecycle will save you hours of rework down the road.

Completed

2. GCD vs. Operations →

← iii. About the Cover