Concurrency in iOS

By Nathan Roberts

A look into how to tackle concepts

of concurrency in iOS development.

Operating Systems

Montana State University

December 4, 2019

Project Files

Below are links to the GitHub repository and the presentation slides for this project.

`https://github.com/nathanroberts7/ios-concurrency`

`https://docs.google.com/presentation/d/1lFWS6Ork3EfUosWffdCV98sFHEUcSFRjaQ2QZBmQ1I0/edit?usp=sharing`

Overview

Concurrency is an important topic to understand as a software engineer. When developing applications, especially those which include a user interface, it is crucial to know how to correctly and effectively run multiple intensive tasks at the same time without hindering the user experience. For iOS development specifically, this report goes over how Apple enables its' developers to work with threads to achieve this goal. Also, a demo is provided to experiment with the concepts mentioned.

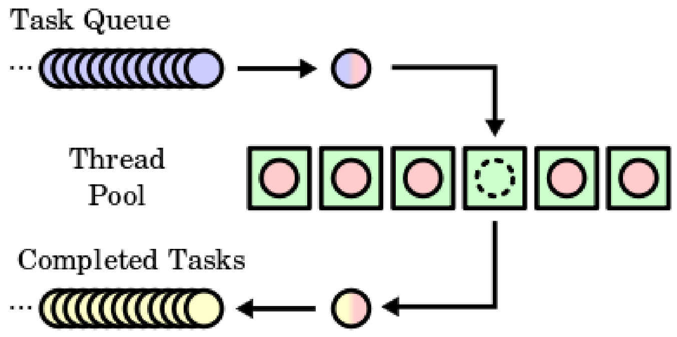
Important Topics

I split this report into several important topics regarding concurrency in iOS development. It will be listed in the same order as the presentation.

1.0.1 Managing Threads via the GCD

Apple realized manually managing threads in your application is difficult, and is very prone to developer mistakes. So, they invented something called the Grand Central Dispatch (GCD), a low-level API that deals with this problem. It implements the thread pool pattern, meaning there are many threads already created and on standby, but not being used. The GCD lets developers exploit a task queue which allows them to interface with a queue, a familiar data structure, and avoid making unseen mistakes. A visualization of this can be seen below in Figure 1.1.

Figure 1.1: Figure of a Thread Pool



The GCD provides developers with four tools: Dispatch Semaphores, Sources, Groups, and Queues. I will explain each one, but we will be focusing on Dispatch

Queues.

Firstly, Dispatch Semaphores are not much different from the regular implementation of a semaphore. It regulates access to a single resource across multiple threads using a simple counter, but, in a very streamlined and efficient way. I've personally never seen these used, so I'll move on to the next tool.

Dispatch Sources let you interact with low level system event signals, file-system events, and other UNIX signals. They basically act as an override for the default implementations, and allow you to run your own custom task instead. I've also never seen these used.

Dispatch Groups let you implement the idea of a critical section in a clean way. They call these critical sections groups, where you enter, leave, and wait for these groups to synchronously complete a task. This is more relevant than the previous two tools mentioned, but similar functionality can be implemented with the last and most used tool: Dispatch Queues.

1.0.2 Dispatch Queues

Dispatch Queues are the go-to when handling concurrency in iOS. They let you execute blocks of code in a variety of ways to achieve your goals of concurrency. There are two primary characteristics of these queues, serial (meaning the queues are executed in the way the were added and run one at a time) and concurrent (meaning they are ran as added, like before, but can run all at once). They also are either synchronous or asynchronous, meaning they either pause execution when ran (until the block of code is complete) or immediately continue execution. The code within each block is guaranteed to run in the order added, so you only need to worry about how and when these Dispatch Queues are being handled.

Lastly, Dispatch Queues have a Quality-of-Service (QoS) attribute which can be quite important when dealing with concurrent queues. QoS is generally used to define priority, and the same applies here. You can define priority for concurrent queues which makes sure a specific concurrent queue is given more resources, but note this does not guarantee order. Generally speaking, however, if each Dispatch Queue you have created contains a relatively large task, the one with the highest QoS should finish first (again, do not rely on this).

1.0.3 Demo Overview

Since this project was a PoC, I have a demo to go along with the use of the GCD! In my demo I go over three main ideas.

1. How to use Dispatch Queues effectively to prevent the UI from freezing
2. Asynchronous versus Synchronous Dispatch Queues, and how this compares to Dispatch Groups
3. How to use the QoS attribute to your advantage with Dispatch Queues

1.0.4 Demo Idea 1: Prevent the UI from Freezing

In iOS, there is only one thread that all the UI is handled on known as the main thread. And, by default, any code you write outside of a dispatch queue is ran on the main thread (although this is not guaranteed). And since you cannot update the UI from any other thread, you definitely want this main thread to be free of any intensive tasks. Otherwise, the user will experience freezes with the user interface.

In the demo, you can try four examples simulating an intense operation. The first two will show how running on the main thread freezes the UI. The third will not freeze the UI, but you will see warnings from updating the UI from another thread besides the main one. The last example is the correct way to run an intense operation and cooperate with the UI without any warnings (using closures). Reference figure 1.2 at the end of the document to get a first-look at the first demo idea.

1.0.5 Demo Idea 2: Asynchronous versus Synchronous

In the second demo idea, I show examples of potential race conditions when using asynchronous queues, and show synchronous queues and dispatch groups as an alternative. Each example will perform a relatively large operation, adding anywhere from one to ten million values to an array. The first asynchronous example will try to reach a certain index before that array is filled, resulting in a race condition. All the other examples are working ones, where the second and third both freeze the UI from synchronous operations. The last example, as before, is a correct implementation of asynchronous queues using closures. Reference figure 1.3 at the end of the document to get a first-look at the second demo idea.

1.0.6 Demo Idea 3: Quality-of-Service

The last demo idea is demonstrating the QoS attribute of Dispatch Queues, showing how it affects the order of completion. And, because of that, I decided to throw in the difference of order between concurrent and serial queues as well. Note, all of the queues used here are asynchronous. For each example, I am performing three relatively quick tasks of setting number to something from 5000 to 5100. Pay attention to the result of the number, and try each example multiple times to see if it is the same. In the first example, I am using serial queues, meaning it will run each queue one at a time from top to bottom. This will result in the same number each time. The second example has concurrent queues, meaning all three are running at the same time. You will notice the number will be different as you keep running the example. The last example uses the QoS attributes, with the queues also being concurrent. The first queue to run has a low priority, and the others have high priority. You will notice since the one with low priority should always finish last, the result stays the same (from the first low priority queue listed). Reference figure 1.4 at the end of the document to get a first-look at the third demo idea.

1.0.7 Relevant Sources

Grand Central Dispatch (GCD):

`https://developer.apple.com/documentation/DISPATCH`

Dispatch Queue:

`https://developer.apple.com/documentation/dispatch/dispatchqueue`

`https://www.swiftbysundell.com/basics/grand-central-dispatch/`

`https://www.appcoda.com/grand-central-dispatch/`

Dispatch Source:

`https://developer.apple.com/documentation/dispatch/dispatchsource`

Dispatch Group:

`https://developer.apple.com/documentation/dispatch/dispatchgroup`

Dispatch Semaphore:

`https://developer.apple.com/documentation/dispatch/dispatchsemaphore`

Figure 1.2: Figure of a Demo Idea 1

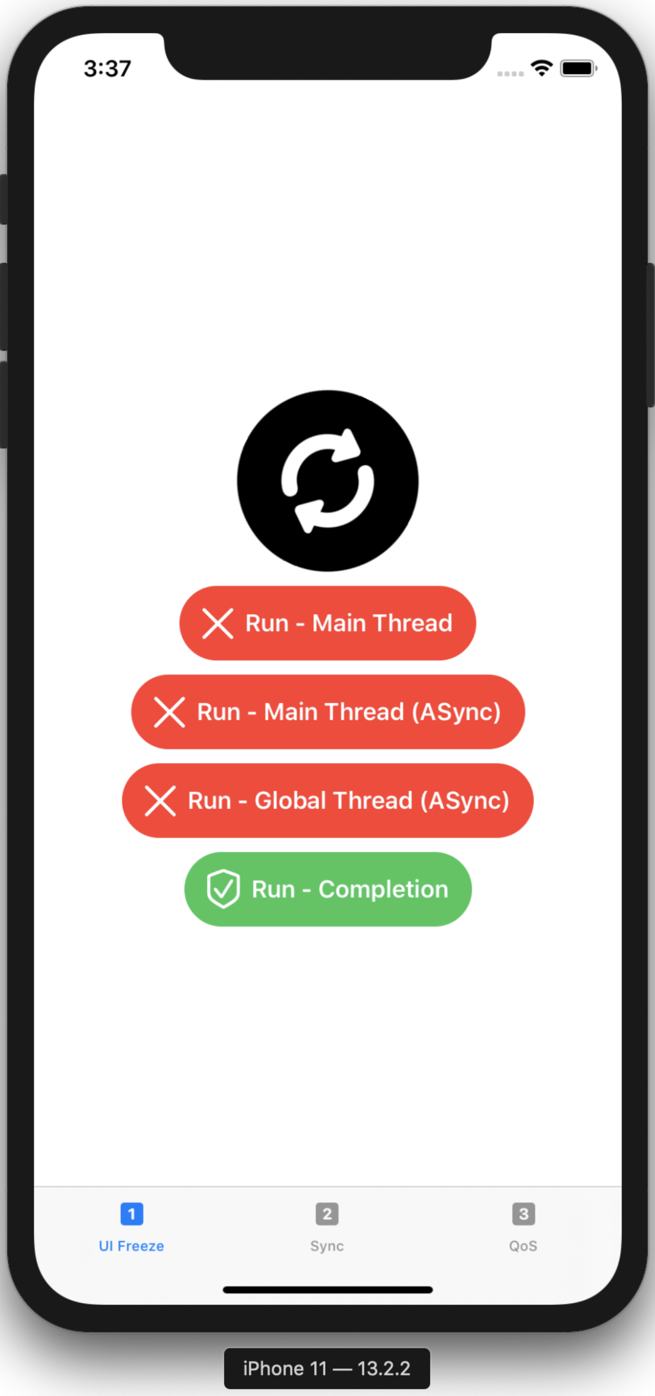


Figure 1.3: Figure of a Demo Idea 2

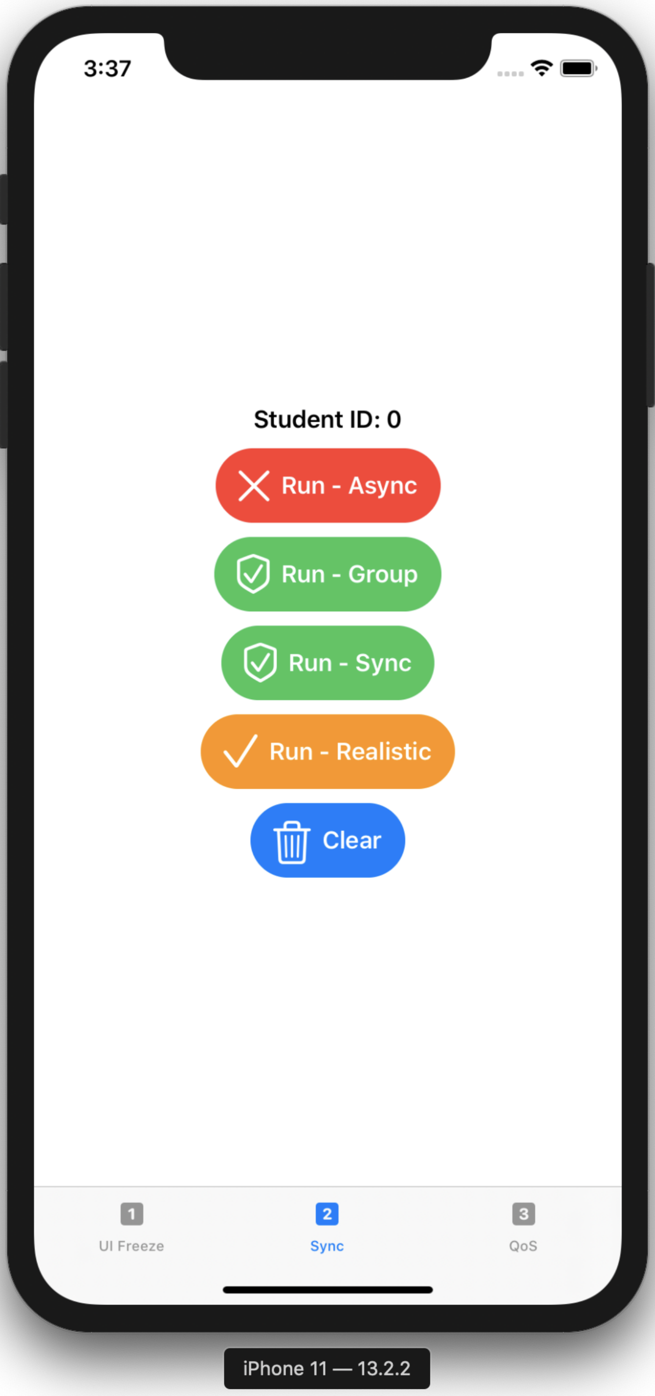
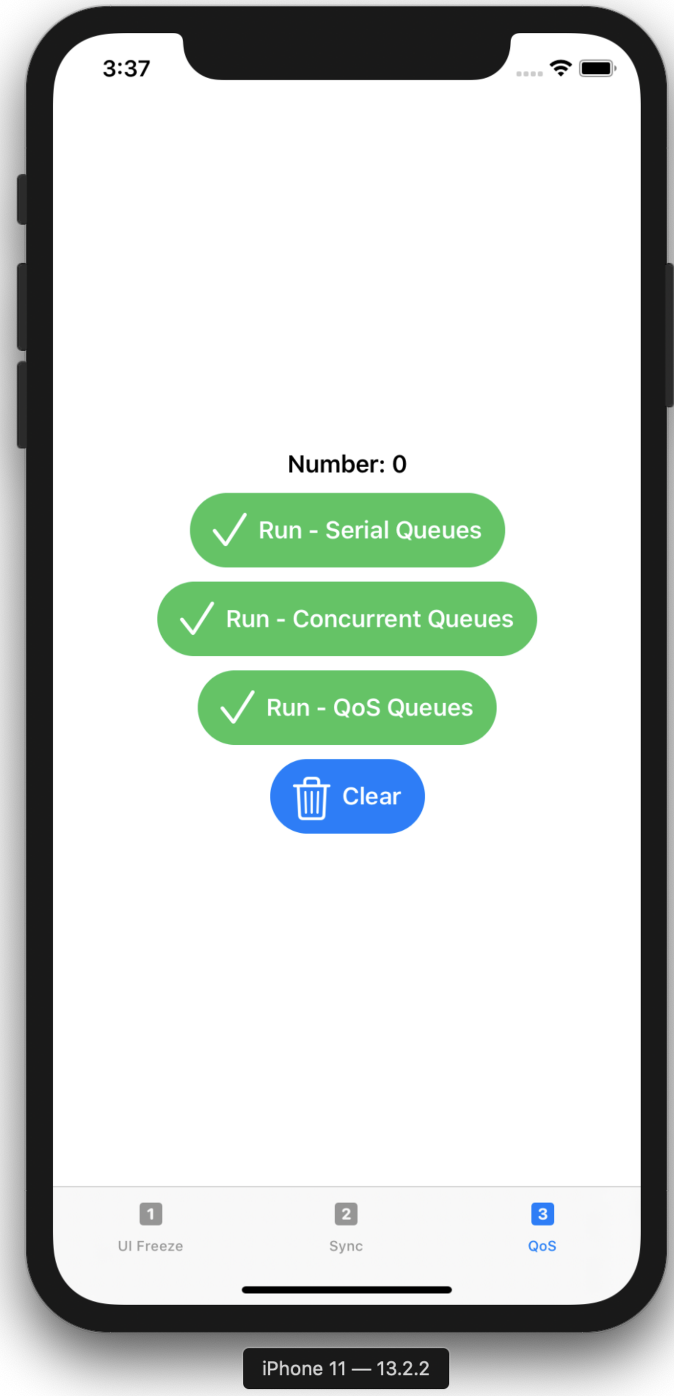


Figure 1.4: Figure of a Demo Idea 3


3:37
Number: 0
Run - Serial Queues
Run - Concurrent Queues
Run - QoS Queues
Clear
1
UI Freeze
2
Sync
3
QoS
iPhone 11 — 13.2.2