**Grand Central Dispatch**

**Grand Central Dispatch(GCD)**

Là một API phổ biến của Apple được sử dụng để hỗ trợ việc xử lý tác vụ đồng thời trên các phần cứng đa lõi của iOS và OSx. Với GCD, chúng ta có thể thực hiện các thao tác sau một cách đơn giản:

* Gửi một tác vụ đến luồng chạy ngầm(background thread) chạy song song với luồng chính(main thread).
* Dùng chung dữ liệu giữa các luồng(mutex).
* Lập lịch chạy cho một tác vụ.
* Xếp các tác vụ chạy theo thứ tự.
* …

GCD hoạt động dựa trên cơ chế Thread Pool: thay vì phải tạo các threads trực tiếp thì chúng ta sẽ đưa các tasks vào queues(hàng đợi). Queues sẽ tự động tạo và huỷ threads, còn việc tạo bao nhiêu threads thì sẽ do hệ thống quyết định dựa trên công việc của CPU đang thực thi. Do đó công việc của developers đơn giản chỉ là quản lý trên Queues mà thôi.

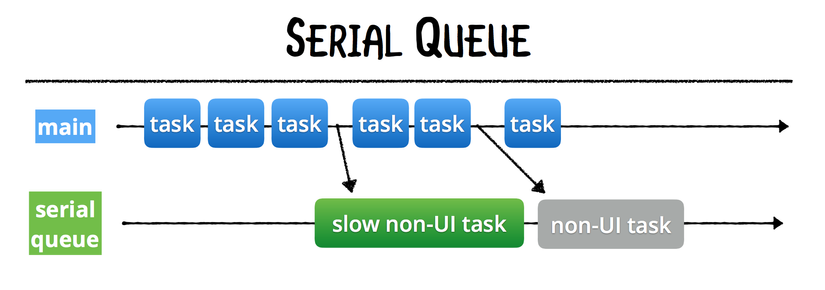
**Queue**

Hoạt động theo nguyên tắc FIFO, task nào vào trước thì sẽ được thực hiện trước, task nào vào sau thì sẽ được thực hiện sau. Có hai loại queues là Serial queue và Concurrent queue.

**Serial queue**

Là hàng đợi thực hiện tuần tự tức là chỉ có 1 task được thực hiện trong 1 thời điểm và khi nào task này thực hiện xong thì task khác mới được bắt đầu.

Main queue là queue đặc biệt có sẵn trong hệ thống và là serial queue.



Hình 1 Minh hoạ về Serial Queue

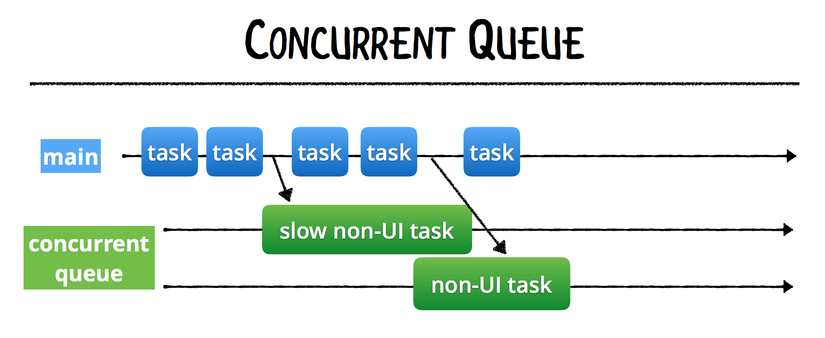
Ở hình minh hoạ tôi khởi tạo serial queue thực hiện một vài tasks, lúc này thì serial queue sẽ tạo ra một thread chạy song song với main thread để thực hiện các tác vụ này.

**Concurrent queue**

Là hàng đợi thực hiện đồng thời. Trong một thời điểm có thể có nhiều task được thực hiện cùng một lúc. Hệ thống sẽ tuỳ vào tải hiện thời của hệ thống và cấu hình phần cứng thực tế để khởi tạo và cấp phát các Threads để xử lý các tác vụ. Global queue là concurrent queue và có sẵn trong hệ thống.

Đến đoạn này chắc nhiều bạn sẽ thắc mắc là Queue hoạt động theo nguyên tắc FIFO thì liệu Concurrent queue có đảm bảo theo nguyên tắc này không?

Câu trả lời là có. Tại sao lại như vậy? Thực chất nguyên tắc này thực hiện trên từng thread và trong concurrent queue thì nó được thực hiện đồng thời trên nhiều threads.

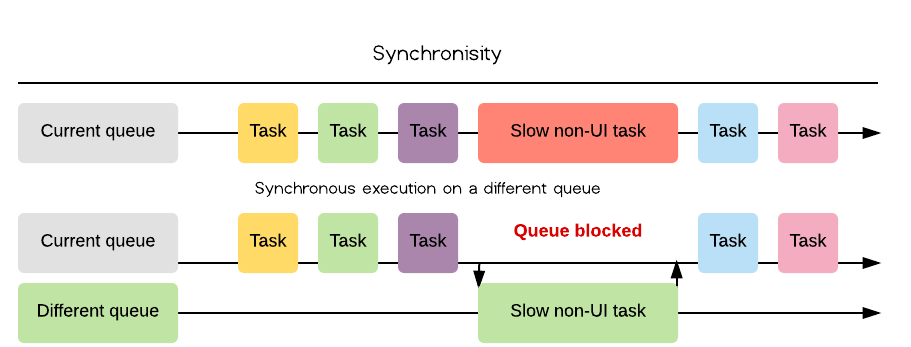


Hình 2 Minh hoạ về Concurrent Queue

**Synchronous và Asynchronous**

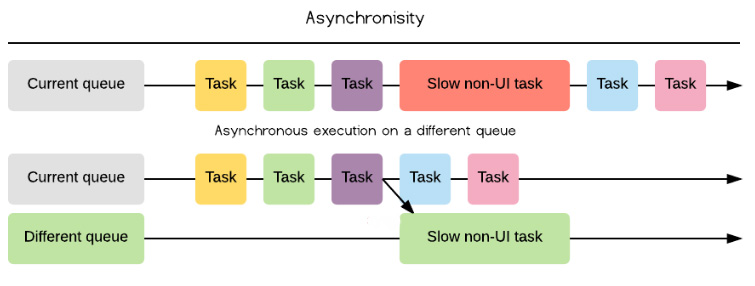
Đầu vào của các queue là các closure hay có thể hiểu nôm na task là một block of code. Các tasks này được đánh dấu về cách thức thực hiện nó trước khi gửi đến một queue. Có hai cách thức thực hiện của một task:

**Synchronous:** Nếu task được đánh dấu là Synchronous(đồng bộ) thì về mặt cơ chế muốn đồng bộ dữ liệu sẽ phải lock các threads trong cùng queue với nó và trả quyền điều khiển cho main queue để thực thi task cần đồng bộ, sau khi task này được thực thi, nó sẽ unlock các threads và đẩy task mới vào.



Hình 3 Minh hoạ về Synchronous

**Asynchronous:** Nếu task được đánh dấu là Asynchronous(bất đồng bộ) thì task này được đẩy vào queue và ngay sau đó nó trả quyền điều khiển cho hàm gọi nó trong khi queue này đang thực thi một task.



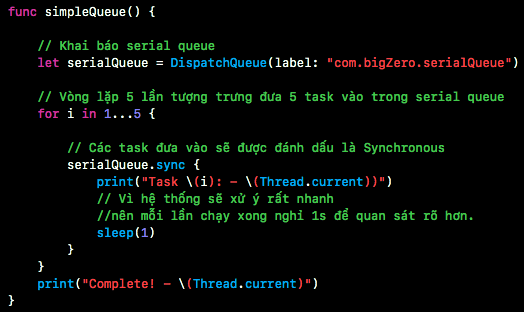
Hình 4 Minh hoạ về Asynchronous

**Lưu ý:** Khi dùng **Serial Queue** và đặc biệt là **Main queue** thì không nên dùng **Synchronous** vì như cơ chế của Sync đã nói ở trên thì trong main queue có một thread duy nhất là main thread, nếu block nó lại thì ai sẽ là người thực thi task đưa vào. Trong khi đó thì main queue(serial queue) sẽ lại đợi cho task được thực hiện xong mới đưa task khác vào dẫn đến main thread sẽ bị lock mãi mãi và hiện tượng này gọi là **deadlock.**

**Demo**

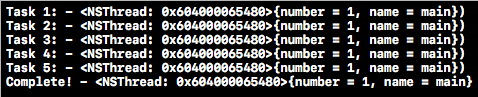
1. **Serial**

* **Gửi sync tasks vào trong serial queue:**



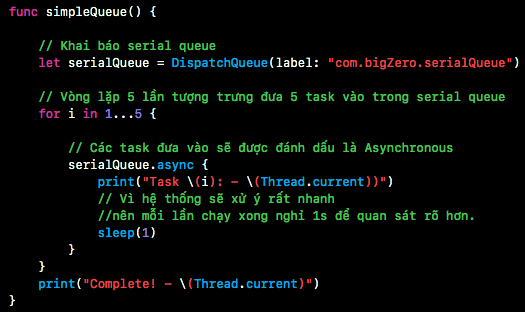
Hình 5 Ví dụ về gửi sync tasks vào Serial queue

Khi gửi một task được đánh dấu là sync vào trong serial queue thì thread mà serial queue đăng kí với hệ thống sẽ bị lock lại và task đưa vào trong queue sẽ được thực thi trên main thread(đồng bộ với main thread). Khi nào task này được thực hiện xong thì thread của serial queue sẽ được unlock và đẩy task mới vào.



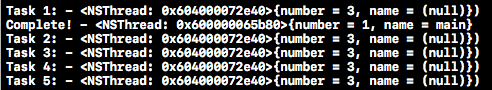
Hình 6 Kết quả xử lý sync tasks trong serial queue

* **Gửi async tasks vào trong serial queue:**



Hình 7 Ví dụ về gửi async tasks vào Serial queue

Khi gửi một task được đánh dấu là async vào trong serial queue thì lúc này thread của serial queue sẽ không bị lock và task này sẽ được thực thi tuần tự ngay trên thread của serial queue và không đồng bộ với main thread.

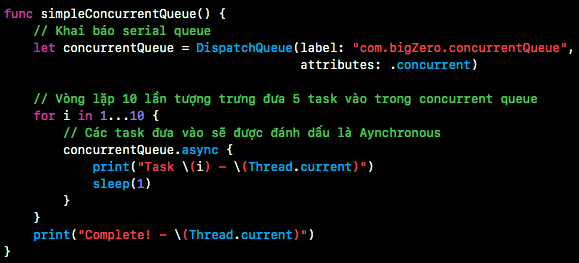


Hình 8 Kết quả xử lý async tasks trong serial queue

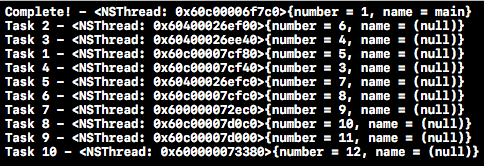
1. **Concurrent**

* **Gửi async tasks vào trong concurrent queue**

Ở ví dụ 10 tasks được đánh dấu là async được gửi vào trong concurrent queue. Queue này sẽ dựa vào tài nguyên hệ thống để tạo ra số threads để thực thi các tasks này.



Hình 9 Ví dụ về gửi async tasks vào Concurrent queue

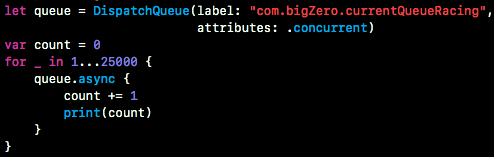


Hình 10 Kết quả xử lý async tasks trong concurrent queue

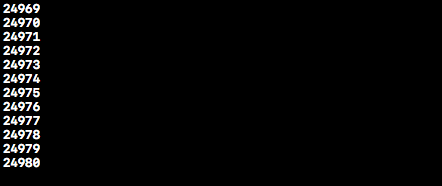
Kết quả concurrent queue tạo ra mười threads mỗi thread thực thi một task và không thằng nào chờ thằng nào.

Tuy nhiên async concurrent queue sử dụng nhiều threads để thực thi nhiểu tasks và bất đồng bộ với nhau nhưng nếu mà các tasks này cùng xử lý trên một tài nguyên thì liệu kết quả cuối cùng có chính xác hay không?

Trường hợp này được gọi là Racing data - trạng thái mà hai hay nhiều luồng (thread) cùng truy xuất, thay đổi tài nguyên dùng chung (shared resource) một cách đồng thời. Tài nguyên dùng chung (shared resource) ở đây có thể là một thuộc tính (property), một object, một file, memory, … Bất cứ mọi tài nguyên dùng chung nào mà được chia sẽ giữa nhiều thread đều tìm ẩn nguy cơ xung đột (conflict).



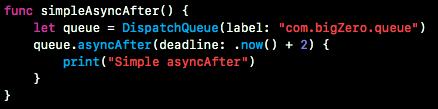
Hình 11 Ví dụ Racing Data



Hình 12 Kết quả ví dụ

Kết quả trả về không chính xác vì các tasks được xử lý bất đồng bộ với nhau, khi 1 thread này đang thực thi công việc cộng thì có một thread khác cũng đang cộng nên dữ liệu không còn được chính xác nữa. Để giải quyết chỉ cần cho các tasks này xử lý đồng bộ (được gọi là safe thread).

**Hẹn giờ cho một task async**

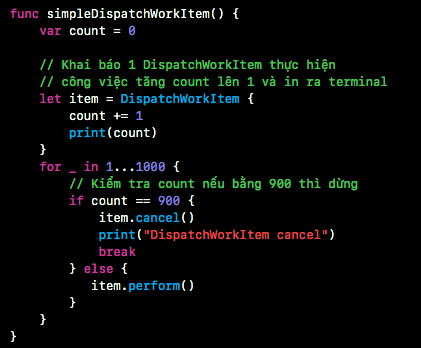
****

Hình 13 Dispatch AsyncAfter

**DispatchWorkItem**

Trong GCD thì để tạm dừng một task đang thực thi thì làm thế nào?

Trước thời điểm iOS 8.0 và macOS 10.10 được ra mắt và DispatchWorkItem được giới thiệu thì điều này bất khả thi với GCD. Nhưng giờ đây DispatchWorkItem chính là chìa khoá cho vần đề này.

****

Hình 14 Ví dụ về DispatchWorkItem

**DispatchGroup**