**Threading and GCD**

Những mục lớn đang chú ý:

* Multithreading
* Queues
* Grand Central Dispathch(GCD)
* Sync (Mã đồng bộ)
* Async (Không đồng bộ)
* DispatchGroup

1. **Multithreading**.

- Ứng dụng sẽ chỉ định 1 Luồng chính để hiển thị giao diện người dùng của bạn và lắng nghe các sự kiện. Các tính toán phức tạp có thể làm chậm luồng chính vậy nên việc đa luồng được hình thành để chuyển tất cả các công việc nặng nhọc sang luồng nền và sau đó chuyển kết quả trở lại tại luồng chính.

Chúng ta có thể tùy chỉnh dược thứ tự ưu tiên của các luồng.

1. **Queues**

- Hàng đợi chỉ là một loạt các khối mã, được xếp thành hàng hoặc xếp hàng đời, chờ một luồng thực thi chúng.

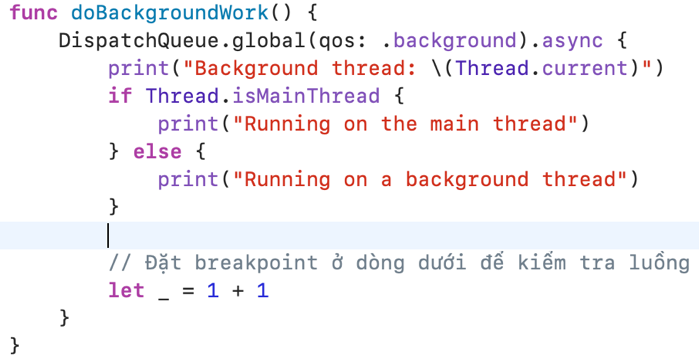
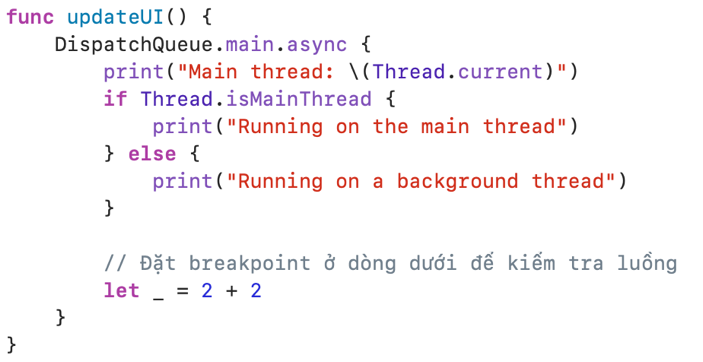
1. **Grand Central Dispathch(GCD)**

**-** Đây là một API giúp quản lý hàng đợi của bạn và trách nhiệm của nó là khởi tạo hàng đợi và đặt các khối mã trên một khối mã đang chờ trong hàng đợi.

+ Có 2 loại hàng đợi

**Main Queue**:(DispatchQueue.main) Hàng đợi nơi tất cả các khối mã ảnh hưởng đến UI phải được chạy trên đó.

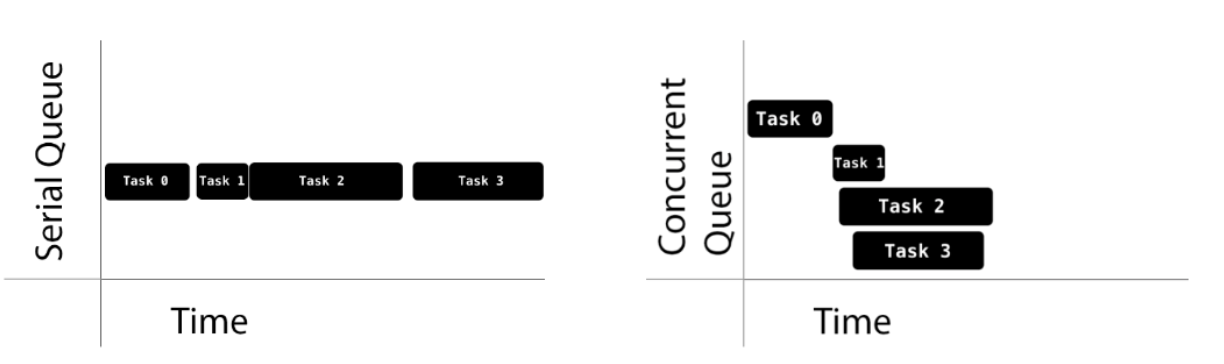
**Background Queue**:(DispatchQueue.global(qos: Qos)) Được sử dụng để xếp hàng các nhiệm vụ tồn tại lâu dài, không phải nhiệm vụ UI, thường chạy đồng thời hoặc song song với hàng đợi UI chính. QOS là độ ưu tiên gồm .userInteractive, .userInitiated, .utility, .background



**Có 2 loại Dispatch Queue**

+ Serial ( Thực thi tuần tự , xử lý xong luồng này rồi mới đến luồng kia )

+ Concurrent ( Thực thi song song) .



**Có 6 loại Qos**

|  |  |
| --- | --- |
| Thứ tự từ cao đến thấp | **Chức năng** |
| **userInteractive** | Mức cao nhất, dùng cho các công việc tương tác trực tiếp với người dùng và cần kết quả ngay lập tức, ví dụ như cập nhật giao diện người dùng. Các công việc ở mức này được thực thi với ưu tiên cao nhất. |
| **userInitiated** | Dùng cho các công việc mà người dùng khởi tạo và mong đợi kết quả trong thời gian ngắn. Ví dụ như mở một tài liệu hay thực hiện một thao tác yêu cầu dữ liệu ngay lập tức. |
| **default** | Mức QoS mặc định cho các công việc không có yêu cầu đặc biệt về hiệu suất. Nếu không xác định QoS, hệ thống sẽ sử dụng mức này. |
| **utility** | Dùng cho các công việc có thời gian thực thi dài, không cần kết quả ngay lập tức, nhưng vẫn quan trọng. Ví dụ như tải về dữ liệu nền hay xử lý các tập tin lớn. |
| **background** | Mức thấp nhất, dùng cho các công việc mà người dùng không cần biết đến và có thể thực hiện trong nền, ví dụ như đồng bộ dữ liệu hay lưu trữ bộ nhớ cache. |
| **unspecified** | Không xác định mức QoS. Đây là mức thấp nhất và không được khuyến nghị sử dụng vì hệ thống không thể tối ưu hóa tài nguyên. |

1. **Synchronous và Asynchronous**

- **Synchronous(Đồng bộ):** các tác vụ được thực thi tuần tự, tức là tác vụ sau chỉ bắt đầu khi tác vụ trước hoàn thành.

- **Asynchronous(Bất đồng bộ):** Tiếp tục thực thi các tác vụ khác mà không chờ đợi, kết quả sẽ được xử lý sau khi hoàn thành.

1. **DispatchGroup**

- Một giải pháp giúp chúng ta có thể thực hiện các tác vụ song song Sau khi tất cả các tác vụ có trong dispatchGroup hoàn thành sẽ tiếp tục tại luồng chính.

Để thêm hàng đợi vào dispatchGroup:

cách 1: Bạn có thể thêm công việc vào DispatchGroup bằng cách sử dụng các phương thức bằng cách sử dụng các phương thức async(group:)

cách 2: cách sử dụng các phương thức **async**(group:)



Chờ đợi các công việc hoàn thành Có hai cách chính để chờ đợi tất cả các công việc trong DispatchGroup hoàn thành: **wait**() và **notify**(queue:).



Lợi ích của DispatchGroup

- **Đồng bộ hóa công việc bất đồng bộ**: **DispatchGroup** giúp bạn dễ dàng quản lý nhiều công việc bất đồng bộ và đảm bảo rằng bạn có thể thực hiện một hành động nào đó sau khi tất cả các công việc đó hoàn thành.

- **Tối ưu hóa hiệu suất**: Bằng cách thực hiện các công việc song song, bạn có thể tận dụng tối đa tài nguyên hệ thống và cải thiện hiệu suất ứng dụng của mình.

- **Dễ sử dụng**: **DispatchGroup** đơn giản và dễ sử dụng với các phương thức trực quan.