

**Truong Minh Thang** @thangtm3003

Theo dõi

★ 986

+ 70

✎ 13



Đăng nhập vào viblo.asia bằng Google

**Dat Vu**

vudat81299@gmail.com

**Dat Vu**

vudat081299@gmail.com

2 tài khoản khác

3.6K 0 7

Concurrency Programming Guide

...



Bài đăng này đã không được cập nhật trong 4 năm



Bài viết này dành cho ai?

Lập trình đồng bộ là một kỹ thuật lập trình trung cấp. Để hiểu được bạn cần phải quen thuộc với các API bất đồng bộ như URLSession, và dễ dàng viết và sử dụng những completion handler closures. Nếu bạn chưa biết những vấn đề trên, bạn cũng có thể xem qua như 1 tài liệu tham khảo, và đừng bắt mình phải hiểu hết vấn đề.

Yếu tố lịch sử

Trong lịch sử phát triển máy tính, khối lượng công việc lớn nhất mà máy tính có thể xử lý trong một đơn vị thời gian được quyết định bởi tốc độ đồng hồ của CPU. Để tăng tốc CPU và thu nhỏ chip bán dẫn, người ta cố gắng nén một lượng lớn các đèn bán dẫn vào trong một diện tích nhỏ nhất. Tuy nhiên cuộc đua tăng tốc cho lõi CPU bị dừng lại do các giới hạn về phần cứng và nhiệt độ. Để tiếp tục tăng tốc cho CPU, người ta bắt đầu tìm một giải pháp khác để tăng tổng hiệu suất của CPU lên, đồng thời tăng hiệu suất tiêu thụ điện. Và giải pháp đó là đưa nhiều lõi hơn vào trong một CPU thay cho một lõi. Việc xử lý được đưa cho nhiều Core cùng xử lý, do đó tổng hiệu năng được tăng lên. Thoạt đầu, Giải pháp nghe có vẻ rất hay nhưng vấn đề lại nằm ở phần mềm. Để tận dụng được lợi thế xử lý nhiều core trong các ứng dụng thì không phải là đơn giản. Trong quá khứ, để sử dụng được các core này, chúng ta phải xử lý việc khởi tạo và quản lý các thread này một cách thủ công. Việc này thực sự khó khăn với hầu hết các lập trình viên, bởi việc xác định được con số tối ưu của các thread trong từng hoàn cảnh dựa trên khối lượng tải hệ thống hiện thời, và phần cứng ở dưới là không hề đơn giản.

Để xử lý vấn đề khó khăn này, cả iOS và OSX đề ra một cách tiếp cận khác cho việc xử lý đồng thời đó là: Thay vì phải tạo các threads một cách trực tiếp, các ứng dụng chỉ đơn giản là gửi các task vào các hàng đợi Queue. Còn việc khởi tạo các thread thế nào, bao nhiêu thread được đẩy cho hệ thống quyết định. Bằng cách để cho hệ thống quản lý quản lý các thread, các ứng dụng có thể đạt được một mức độ linh hoạt mà cách xử lý cũ không bao giờ đạt được. Đồng thời lập trình viên có được một mô hình lập trình đơn giản mà hiệu quả hơn.

Concurrency là gì?

Xử lý đồng thời - Concurrency - là việc nhiều task được xử lý cùng một lúc.

Tại sao app của chúng ta lại cần xử lý đồng thời -



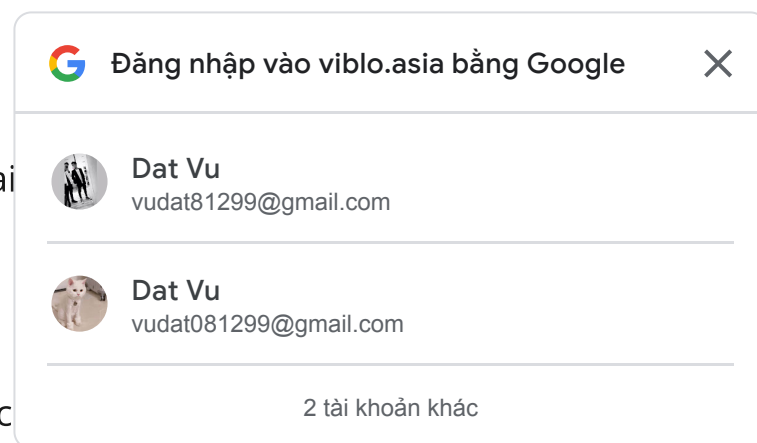
↑ +19 ↓



- Để giữ cho UI luôn trong trạng thái được đáp ứng.
- Tăng tốc độ xử lý, tận dụng tối đa sức mạnh của kiến trúc chip đa nhân.

Nếu chúng ta xử lý một task non-UI nặng trên main thread, task này sẽ block lại không thể tiếp nhận được những tương tác của người dùng nữa.

Lúc này chúng ta cần chuyển các tác vụ nặng non-UI task sang một thread khác làm các nhiệm vụ khác trong đó có nhiệm vụ quan trọng nhất là đón nhận những tương tác của người dùng.



Một số khái niệm của lập trình đồng thời

Concurrency

Concurrency không chỉ là một khái niệm cho thiết bị có chip nhiều nhân. Trong những thiết bị đơn nhân, chúng ta vẫn có thể xử lý được đa luồng dựa vào cơ chế time-slicing để chuyển ngữ cảnh.

Queue

Queue là hàng đợi các công việc, hoạt động theo nguyên tắc FIFO, task nào vào trước thì sẽ được thực hiện trước, task nào vào sau sẽ được thực hiện sau. Có hai loại hàng đợi: **Serial Queue**: là hàng đợi thực hiện theo tuần tự. Trong một thời điểm chỉ có 1 task được thực thi. Khi nào task này thực thi xong thì task khác mới bắt đầu. Ví dụ tiêu biểu của hàng đợi này là Main thread.

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 tùy vào tải hiện thời của hệ thống và cấu hình phần cứng thực tế để khởi tạo và cấp phát các Thread để xử lý các tác vụ.

So sánh giữa Serial Queue và Concurrent Queue

Synchronous và Asynchronous

Đầu vào của các queue là các closure. Các closure này được đánh dấu về cách thức thực hiện nó trước khi gửi đến một queue. Có hai cách thức thực hiện của một closure: Nếu task đánh dấu là Synchronous thì task này sẽ block lại queue mà nó được gọi, không cho phép queue đó thực thi thêm task nào khác trong thời gian nó đang chạy. Nếu task được đánh dấu là Asynchronous thì task này được gọi và ngay sau đó nó trả quyền điều khiển cho hàm gọi nó và hàng đợi sẽ thực thi một closure tiếp theo (nếu có đủ queue để thực thi).

Mối quan hệ giữa Synchronous, Asynchronous VS Serial Queue, Concurrent Queue

Synchronous, Asynchronous là cách thức thực hiện của 1 task. Serial Queue, Concurrent Queue là đích đến của task đó.

Synchronous, Asynchronous nói cho bạn biết là queue hiện thời có phải đợi task hoàn thành rồi mới gọi task mới hay không. Serial Queue, Concurrent Queue thì cho bạn biết là với queue hiện thời, bạn có 1 thread hay nhiều thread. 1 Task được thực hiện 1 lúc hay nhiều task được thực hiện đồng thời.



↑ +19 ↓



Trường hợp gửi 2 async task vào serial queue


```
func simpleQueues() {
    let queue = DispatchQueue(label: "com.bigZero.GCDSamples")


    queue.async {
        for i in 0..<5 {
            print("🟦 \((i) - \(( Thread.current))")
        }
    }

    queue.async {
        for i in 0..<5 {
            print("🏟️ \((i) - \((Thread.current))")
        }
    }

    for i in 0..<10 {
        print("♥️ \((i) - \((Thread.current))")
    }
}
```

Đăng nhập vào viblo.asia bằng Google

 **Dat Vu**
vudat81299@gmail.com

 **Dat Vu**
vudat081299@gmail.com

2 tài khoản khác

khi gửi một async task in 🟦 vào trong queue, ngay lập tức nó trả quyền điều khiển cho function gọi nó. Vì vậy chúng ta tiếp tục chạy được dòng lệnh gửi một async task thứ 2 vào trong queue - task in 🏟️ async task thứ 2 sau khi được đẩy vào trong Queue, nó ngay lập tức trả điều khiển lại cho function gọi nó và function simpleQueues() tiếp tục thực thi việc in ♥️ Do task 🟦 và task 🏟️ được đưa vào cùng 1 Serial Queue nên nó được chạy trên 1 thread theo cách thức tuần tự. Task ♥️ được thực hiện trên thread hiện thời là main thread. (🟦 tuần tự 🏟️) // Main thread Task Vì main thread có mức độ ưu tiên cao nhất nên trong quá trình thực hiện, mặc dù số ♥️ bằng tổng số 🟦 + 🏟️. Nhưng khi thực hiện ♥️ vẫn thực hiện xong trước 2 task kia


```
🟦 0 -<NSThread: 0x610000078200>{number = 3, name = (null)}
♥️ 0 - <NSThread: 0x610000070d80>{number = 1, name = main}
♥️ 1 - <NSThread: 0x610000070d80>{number = 1, name = main}
🟦 1 -<NSThread: 0x610000078200>{number = 3, name = (null)}
♥️ 2 - <NSThread: 0x610000070d80>{number = 1, name = main}
🟦 2 -<NSThread: 0x610000078200>{number = 3, name = (null)}
♥️ 3 - <NSThread: 0x610000070d80>{number = 1, name = main}
♥️ 4 - <NSThread: 0x610000070d80>{number = 1, name = main}
🟦 3 -<NSThread: 0x610000078200>{number = 3, name = (null)}
♥️ 5 - <NSThread: 0x610000070d80>{number = 1, name = main}
🟦 4 -<NSThread: 0x610000078200>{number = 3, name = (null)}
♥️ 6 - <NSThread: 0x610000070d80>{number = 1, name = main}
♥️ 7 - <NSThread: 0x610000070d80>{number = 1, name = main}
🏟️ 0 - <NSThread: 0x610000078200>{number = 3, name = (null)}
♥️ 8 - <NSThread: 0x610000070d80>{number = 1, name = main}
♥️ 9 - <NSThread: 0x610000070d80>{number = 1, name = main}
🏟️ 1 - <NSThread: 0x610000078200>{number = 3, name = (null)}
🏟️ 2 - <NSThread: 0x610000078200>{number = 3, name = (null)}
🏟️ 3 - <NSThread: 0x610000078200>{number = 3, name = (null)}
🏟️ 4 - <NSThread: 0x610000078200>{number = 3, name = (null)}
```


Trường hợp gửi 2 task Sync vào Serial Queue


```
func simpleQueues() {
    let serialQueue = DispatchQueue(label: "com.bigZero.GCDSamples")
    serialQueue.sync {
        for i in 0..<5 {
            print("🟦 \(i) - \( Thread.current)")
        }
    }

    serialQueue.sync {
        for i in 0..<5 {
            print("🏏 \(i) - \(Thread.current)")
        }
    }

    for i in 0..<10 {
        print("♥ \(i) - \(Thread.current)")
    }
}
```


 Đăng nhập vào viblo.asia bằng Google
 ✕


Dat Vu
 vudat81299@gmail.com


Dat Vu
 vudat081299@gmail.com

2 tài khoản khác

Khi gửi sync task 🟦 vào serialQueue, task 🟦 block lại function simpleQueues không cho thực hiện tiếp các tác vụ tiếp theo. Lúc này main thread sẽ được rảnh (vì main queue đang bị block lại), vì vậy nó được cấp phát để thực hiện task 🟦. Sau khi thực hiện xong task 🟦, task 🏏 bắt đầu được gọi và cũng tiếp tục như trên. cuối cùng task ♥ được gọi trên main queue, và với tất cả các task vụ trên main queue đều sẽ được ưu tiên thực hiện trên main thread.

```
🟦 0 - <NSThread: 0x610000073200>{number = 1, name = main}
🟦 1 - <NSThread: 0x610000073200>{number = 1, name = main}
🟦 2 - <NSThread: 0x610000073200>{number = 1, name = main}
🟦 3 - <NSThread: 0x610000073200>{number = 1, name = main}
🟦 4 - <NSThread: 0x610000073200>{number = 1, name = main}
🏏 0 - <NSThread: 0x610000073200>{number = 1, name = main}
🏏 1 - <NSThread: 0x610000073200>{number = 1, name = main}
🏏 2 - <NSThread: 0x610000073200>{number = 1, name = main}
🏏 3 - <NSThread: 0x610000073200>{number = 1, name = main}
🏏 4 - <NSThread: 0x610000073200>{number = 1, name = main}
♥ 0 - <NSThread: 0x610000073200>{number = 1, name = main}
♥ 1 - <NSThread: 0x610000073200>{number = 1, name = main}
♥ 2 - <NSThread: 0x610000073200>{number = 1, name = main}
♥ 3 - <NSThread: 0x610000073200>{number = 1, name = main}
♥ 4 - <NSThread: 0x610000073200>{number = 1, name = main}
♥ 5 - <NSThread: 0x610000073200>{number = 1, name = main}
♥ 6 - <NSThread: 0x610000073200>{number = 1, name = main}
♥ 7 - <NSThread: 0x610000073200>{number = 1, name = main}
♥ 8 - <NSThread: 0x610000073200>{number = 1, name = main}
♥ 9 - <NSThread: 0x610000073200>{number = 1, name = main}
```

Bây giờ chúng ta thay đổi 1 chút, task thứ 2 chúng ta chuyển thành async

```
func simpleQueues() {
    let serialQueue = DispatchQueue(label: "com.bigZero.GCDSamples")
    serialQueue.sync {
        for i in 0..<5 {
            print("🟦 \(i) - \( Thread.current)")
        }
    }

    serialQueue.async {
        for i in 0..<5 {
            print("🏏 \(i) - \(Thread.current)")
        }
    }

    for i in 0..<10 {
        print("♥ \(i) - \(Thread.current)")
    }
}
```



↑ +19 ↓




```
}
```

cũng như ở trên Khi gửi sync task 🟦 vào serialQueue, task 🟦 block lại function s tiếp các tác vụ tiếp theo. Lúc này main thread sẽ được rảnh (vì main queue đang phát để thực hiện task 🟦. Sau khi thực hiện xong task 🟦, task 🏏 bắt đầu được g tức nó trả lại quyền điều khiển cho function gọi nó và task ❤️ được gọi ưu tiên tr đang được sử dụng rồi, nên hệ thống cấp phát cho task 🏏 một thread khác để

```
🟦 0 -<NSThread: 0x6100000757c0>{number = 1, name = main}
🟦 1 -<NSThread: 0x6100000757c0>{number = 1, name = main}
🟦 2 -<NSThread: 0x6100000757c0>{number = 1, name = main}
🟦 3 -<NSThread: 0x6100000757c0>{number = 1, name = main}
🟦 4 -<NSThread: 0x6100000757c0>{number = 1, name = main}
🏏 0 - <NSThread: 0x60800007a980>{number = 4, name = (null)}
♥ 0 - <NSThread: 0x6100000757c0>{number = 1, name = main}
🏏 1 - <NSThread: 0x60800007a980>{number = 4, name = (null)}
♥ 1 - <NSThread: 0x6100000757c0>{number = 1, name = main}
♥ 2 - <NSThread: 0x6100000757c0>{number = 1, name = main}
🏏 2 - <NSThread: 0x60800007a980>{number = 4, name = (null)}
♥ 3 - <NSThread: 0x6100000757c0>{number = 1, name = main}
♥ 4 - <NSThread: 0x6100000757c0>{number = 1, name = main}
🏏 3 - <NSThread: 0x60800007a980>{number = 4, name = (null)}
♥ 5 - <NSThread: 0x6100000757c0>{number = 1, name = main}
🏏 4 - <NSThread: 0x60800007a980>{number = 4, name = (null)}
♥ 6 - <NSThread: 0x6100000757c0>{number = 1, name = main}
♥ 7 - <NSThread: 0x6100000757c0>{number = 1, name = main}
♥ 8 - <NSThread: 0x6100000757c0>{number = 1, name = main}
♥ 9 - <NSThread: 0x6100000757c0>{number = 1, name = main}
```

Trường hợp gửi 3 task Async vào Concurrent Queue


```
func concurrentQueues() {
    let concurrentQueue = DispatchQueue.global()
    concurrentQueue.async {
        for i in 0..<10 {
            print("🟦 \(i) - \(Thread.current)")
        }
    }

    concurrentQueue.async {
        for i in 0..<10 {
            print("♥ \(i) - \(Thread.current)")
        }
    }


    concurrentQueue.async {
        for i in 0..<10 {
            print("🏏 \(i) - \(Thread.current)")
        }
    }
}
```

Do 3 task đều là async nên cả 3 task đều được đưa vào trong concurrentQueue. Lúc này hệ thống sẽ cấp phát cho concurrentQueue 3 thread để thực hiện đồng thời 3 task trên 3 thread khác nhau do đó ta được

Đăng nhập vào viblo.asia bằng Google




Dat Vu
vudat81299@gmail.com




Dat Vu
vudat081299@gmail.com

2 tài khoản khác

Đăng nhập vào viblo.asia bằng Google

Dat Vu
vudat81299@gmail.com

Dat Vu
vudat081299@gmail.com


2 tài khoản khác


```
0 - <NSThread: 0x600000069480>{number = 3, name = (null)}
♥ 0- <NSThread: 0x600000069500>{number = 5, name = (null)}
0- <NSThread: 0x608000066880>{number = 1, name = main}
1 - <NSThread: 0x600000069480>{number = 3, name = (null)}
1- <NSThread: 0x608000066880>{number = 1, name = main}
♥ 1- <NSThread: 0x600000069500>{number = 5, name = (null)}
2- <NSThread: 0x608000066880>{number = 1, name = main}
2 - <NSThread: 0x600000069480>{number = 3, name = (null)}
♥ 2- <NSThread: 0x600000069500>{number = 5, name = (null)}
3- <NSThread: 0x608000066880>{number = 1, name = main}
3 - <NSThread: 0x600000069480>{number = 3, name = (null)}
♥ 3- <NSThread: 0x600000069500>{number = 5, name = (null)}
4- <NSThread: 0x608000066880>{number = 1, name = main}
4 - <NSThread: 0x600000069480>{number = 3, name = (null)}
♥ 4- <NSThread: 0x600000069500>{number = 5, name = (null)}
5- <NSThread: 0x608000066880>{number = 1, name = main}
5 - <NSThread: 0x600000069480>{number = 3, name = (null)}
♥ 5- <NSThread: 0x600000069500>{number = 5, name = (null)}
6- <NSThread: 0x608000066880>{number = 1, name = main}
6 - <NSThread: 0x600000069480>{number = 3, name = (null)}
♥ 6- <NSThread: 0x600000069500>{number = 5, name = (null)}
7- <NSThread: 0x608000066880>{number = 1, name = main}
7 - <NSThread: 0x600000069480>{number = 3, name = (null)}
♥ 7- <NSThread: 0x600000069500>{number = 5, name = (null)}
8- <NSThread: 0x608000066880>{number = 1, name = main}
8 - <NSThread: 0x600000069480>{number = 3, name = (null)}
♥ 8- <NSThread: 0x600000069500>{number = 5, name = (null)}
9- <NSThread: 0x608000066880>{number = 1, name = main}
9 - <NSThread: 0x600000069480>{number = 3, name = (null)}
♥ 9- <NSThread: 0x600000069500>{number = 5, name = (null)}
```

Thay đổi một chút. Ta cho task ♥ trở thành sync. Khi add xong task ♥, do task này là sync nên nó khoá queue lại, không cho add task vào nữa. Sau khi task ♥ chạy xong, task mới được add vào concurrent Queue. Do vậy như kết quả, task chạy 1 mình cuối cùng.


```
♥ 4- <NSThread: 0x610000065980>{number = 1, name = main}
4 - <NSThread: 0x618000064380>{number = 3, name = (null)}
♥ 5- <NSThread: 0x610000065980>{number = 1, name = main}
5 - <NSThread: 0x618000064380>{number = 3, name = (null)}
♥ 6- <NSThread: 0x610000065980>{number = 1, name = main}
♥ 7- <NSThread: 0x610000065980>{number = 1, name = main}
6 - <NSThread: 0x618000064380>{number = 3, name = (null)}
♥ 8- <NSThread: 0x610000065980>{number = 1, name = main}
7 - <NSThread: 0x618000064380>{number = 3, name = (null)}
♥ 9- <NSThread: 0x610000065980>{number = 1, name = main}
8 - <NSThread: 0x618000064380>{number = 3, name = (null)}
9 - <NSThread: 0x618000064380>{number = 3, name = (null)}
0- <NSThread: 0x618000064380>{number = 3, name = (null)}
1- <NSThread: 0x618000064380>{number = 3, name = (null)}
2- <NSThread: 0x618000064380>{number = 3, name = (null)}
3- <NSThread: 0x618000064380>{number = 3, name = (null)}
4- <NSThread: 0x618000064380>{number = 3, name = (null)}
5- <NSThread: 0x618000064380>{number = 3, name = (null)}
6- <NSThread: 0x618000064380>{number = 3, name = (null)}
7- <NSThread: 0x618000064380>{number = 3, name = (null)}
8- <NSThread: 0x618000064380>{number = 3, name = (null)}
9- <NSThread: 0x618000064380>{number = 3, name = (null)}
```

Một lưu ý cuối cùng, chúng ta tạo queue, chúng ta có thể quyết định rằng queue này sẽ chạy trên 1 thread hay nhiều thread bằng cách chỉ định loại queue là Serial hay Concurrent. Nhưng chính xác là thread nào thực thi các tác vụ thì chúng ta ko được quyền quyết định, việc đó được đẩy cho hệ thống quyết định dựa trên các yếu tố phần

 Đăng nhập vào viblo.asia bằng Google



Dat Vu
vudat81299@gmail.com



Dat Vu
vudat081299@gmail.com

2 tài khoản khác

All rights reserved

Bài viết liên quan

Multithreading: Các cách khởi tạo và sử dụng Java Thread

[Cùi Bắp](#)
4 phút đọc
👁 10599 📌 6 💬 2 ⬆ 5

Làm quen với Multithreading Trong C++

[Vu Trung Kien](#)
6 phút đọc
👁 14728 📌 6 💬 0 ⬆ 3

Phần 5: Cách bảo mật giao dịch Tiền ảo bằng Ví tiền ảo

[Chung Minh Tú](#)
5 phút đọc
👁 2022 📌 3 💬 2 ⬆ 10

Phần 4: Thử trao đổi tiền MyCoin trong mạng ngang hàng bằng Blockchain

[Chung Minh Tú](#)
4 phút đọc
👁 2633 📌 2 💬 12 ⬆ 8

Bài viết khác từ Truong Minh Thang

Function và block là một class?

[Truong Minh Thang](#)
1 phút đọc
👁 423 📌 1 💬 0 ⬆ 3

Funtion tiến hoá trở thành Closure và cái kết bất ngờ!

[Truong Minh Thang](#)
3 phút đọc
👁 1059 📌 3 💬 0 ⬆ 9

Reference Type (class)_VS_ Value Type (enum, struct).

[Truong Minh Thang](#)
4 phút đọc
👁 1333 📌 7 💬 0 ⬆ 12

Một số lưu ý khi đặt tên hàm tên biến theo chuẩn của apple

[Truong Minh Thang](#)
8 phút đọc
👁 2736 📌 4 💬 1 ⬆ 3


Bình luận






💬 Đăng nhập để bình luận

TÀI NGUYÊN

DỊCH VỤ

ỨNG DỤNG DI ĐỘNG

 ↑ +19 ↓

 ·  ·  ·  · 

[Câu hỏi](#)

[Tags](#)

[CV](#) [Viblo CV](#)

[Videos](#)

[Tác giả](#)

[CTF](#) [Viblo CTF](#)

[Thảo luận](#)

[Đề xuất hệ thống](#)

[Viblo Learning](#)

[Công cụ](#)

[Machine Learning](#)

[Trạng thái hệ thống](#)

 Tiếng Việt 

[Về chúng tôi](#)

[Phản hồi](#)

[Giúp đỡ](#)

[FAQs](#)


[RSS](#)

[Điều khoản](#)


[DMCA](#)  [PROTECTED](#)

© Viblo 2021

 Download on the
App Store

 Đăng nhập vào viblo.asia bằng Google 

 **Dat Vu**
vudat81299@gmail.com

 **Dat Vu**
vudat081299@gmail.com

2 tài khoản khác



↑ +19 ↓

