Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

Lập trình iOS

Bài 4. *Tiến trình và xử lý đa tiến trình*

Ngành Mạng & Thiết bị di động

2015

Nội dung

1. **Tiến trình**
   - Giới thiệu NSThread
   - Cách sử dụng NSThread
2. **Định nghĩa tầm quan trọng của Multithreading**
3. **Multithreading trong Objective - C**

1.1 Giới thiệu NSThread

- Một đối tượng NSThread quản lý một tiến trình (thread) được thực hiện.
- NSThread được sự dụng trong trường hợp muốn tạo ra một tiến trình để thực hiện một nhiệm vụ nào đó song song với tiến trình chính (main thread).
- Tránh làm cho main thread bị block, xử lý các công việc liên quan tới giao diện người dùng, các hoạt động liên quan đến sự kiện.
- Tiến trình còn dùng để phân chia các công việc lớn thành các công việc nhỏ hơn khai thác tối đa hiệu suất trên các thiết bị đa lõi.

1.2 Cách sử dụng NSThread

- **Khởi tạo một NSThread**
  - **init**: Đây là phương thức khởi tạo một NSThread. Giá trị trả về là một giá trị NSThread được khởi tạo.
  - **initWithTarget:selector:object**: Phương thức tạo một tiến trình với mục tiêu thực hiện là một phương thức với đối số truyền vào. Các ứng dụng không sử dụng ARC phải sử dụng thêm autorelease pool. Các ứng dụng sử dụng ARC không cần sử dụng autorelease pool. Đối tượng target và các đối số sẽ được khởi tạo và sử dụng trong tiến trình cho tới khi tiến trình kết thúc.

1.2 Cách sử dụng NSThread

- **Bắt đầu và dừng tiến trình**
  - + **detachNewThreadSelector:toTarget:withObject**: phương thức này giúp lấy riêng ra một tiến trình mới và xử lý theo phương thức selector.
  - - **start**: phương thức bắt đầu chạy tiến trình.
  - + **sleepUntilDate**: dừng tiến trình cho tới một thời gian quy định.
  - + **sleepForTimeInterval**: Dừng tiến trình trong một khoảng thời gian quy định.
  - + **exit**: Chấm dứt hoạt động của tiến trình. Trước khi thoát phương thức này sẽ gửi một NSThreadWillExitNotification đối với các tiến trình được thoát tới trung tâm notification mặc định.
  - - **cancel**: Thay đổi trạng thái hủy của tiến trình và thoát ra. Để có thể kiểm tra một tiến trình có ở trạng thái cancel hay không, ta có thể sử dụng phương thức isCancelled.

1.2 Cách sử dụng NSThread

- **Xác định trạng thái của một tiến trình:**
  - **executing**: thuộc tính cho biết tiến trình có thực hiện hay chưa.
  - **finished**: thuộc tính cho biết tiến trình đã hoàn thành hay chưa.
  - **cancelled**: thuộc tính cho biết tiến trình có bị hủy bỏ hay không.

- **Làm việc với main thread:**
  - + **isMainThread**: phương thức trả về giá trị cho biết tiến trình đang thực hiện có phải là main thread hay không.
  - **isMainThread**: thuộc tính cho biết tiến trình đang thực hiện có phải là main thread hay không. Thuộc tính này chỉ cho phép đọc.
  - **+ mainThread**: Trả về tiến trình đại diện cho main thread.

1.2 Cách sử dụng NSThread

- **Truy vấn môi trường**
  - + **isMultiThreaded**: Kiểm tra xem ứng dụng hiện tại có sử dụng đa luồng hay không. Nếu có là YES ngược lại là NO.
  - + **currentThread**: Trả về tiến trình hiện tại của chương trình đang được thực hiện.

- **Làm việc với độ ưu tiên của tiến trình:**
  - + **threadPriority** : Trả về giá trị ưu tiên của tiến trình đang hiện hành. Với 1.0 là ưu tiên cao nhất và 0.0 là thấp nhất.
  - **threadPriority** : Thuộc tính lưu trữ giá trị ưu tiên của đối tượng được nhận. Giá trị sẽ có từ 0.0 – 1.0 với 1.0 là lớn nhất.

Nội dung

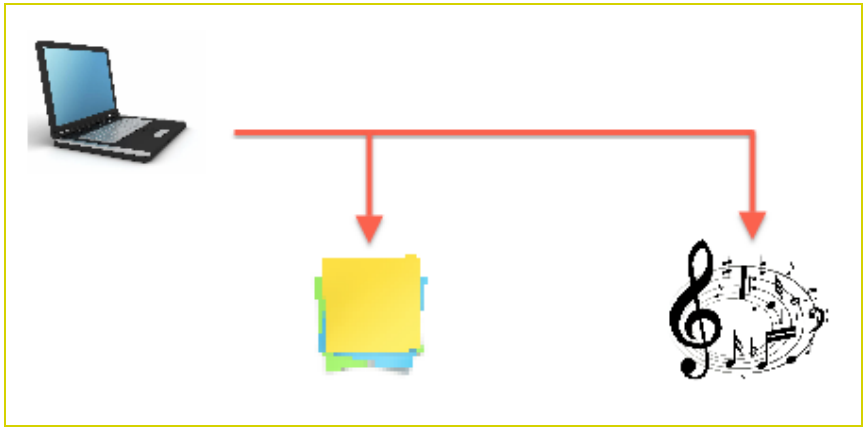
1. Tiến trình

2. **Định nghĩa tầm quan trọng của Multithreading**
   - Định nghĩa Multithreading
     - Multitasking (đa nhiệm)
     - Multithreading (đa luồng)
   - Ý nghĩa và ví dụ

3. Multithreading trong Objective - C

2.1 Định nghĩa Multithreading

- **Multitasking (đa nhiệm)**
  - Multitasking là thực thi hai hay nhiều tác nhiệm cùng một lúc. Gần như tất cả các hệ điều hành đều có khả năng multitasking và sử dụng một trong hai kỹ thuật là : multitasking dựa trên process(xử lý) hay multitasking dựa trên tiến trình (phân tuyến).
    - Multitasking dựa trên process là chạy hai chương trình cùng một lúc.
    - Multitasking dựa trên thread là có một chương trình thực hiện hai hay nhiều tác nhiệm tại cùng một thời điểm.



2.1 Định nghĩa Multithreading

- **Multitasking (đa nhiệm)**
  - Mục tiêu của multitasking là tận dụng thời gian nghĩ của CPU. Cũng giống như việc bạn vừa nghe nhạc và làm việc chẳng hạn. Trong quá trình làm việc có những lúc chúng ta không hoạt động, hay không suy nghĩ thì chúng ta sẽ nghe nhạc.
  - Việc nghe nhạc xảy ra đồng thời song song với công việc trong tình huống ấy chúng ta chính là CPU.
  - Việc sử dụng multitasking cũng giống như việc chúng ta muốn CPU của bạn xoay vòng để xử lý các lệnh và dữ liệu và đạt được hiệu quả một cách tốt nhất.

2.1 Định nghĩa Multithreading

- **Multithreading (đa luồng)**
  - Multithreading chính là multitasking thread như định nghĩa ở trên. Multithreading là hai hoặc nhiều phần của chương trình được chạy đồng thời cùng một lúc và mỗi phần là một tiến trình.

2.2 Ý nghĩa và ví dụ

- Vì tận dụng được CPU của máy. Nên về sử dụng multithreading sẽ giúp bạn điều khiển các xử lý đòi hỏi tính toán nhanh hơn.
- Đặc biệt trong trường hợp tốc độ truyền tải dữ diệu là chậm hoặc trong trường hợp xử lý các dữ liệu lớn, xử lý các hành động phức tạp cần thời gian.
- Xử lý các thao tác đồng thời nhằm giảm thiểu thời gian và đạt hiểu qua tốt đa.

2.2 Ý nghĩa và ví dụ

- **Ví dụ cụ thể : Trong trường hợp ứng dụng cần load số lượng hình ảnh lớn từ server và hiển thị. Trong trường hợp này multithreading sẽ giúp hành động load dữ liệu hình ảnh và hiển thị diễn ra đồng thời. Giúp cho ứng dụng không bị block (khóa).**

Nội dung

1. Tiến trình

2. Định nghĩa tầm quan trọng của Multithreading

3. **Multithreading trong Objective – C**
   - Operation Queues
   - Dispatch Queues
   - Dispatch Sources

Hàng đợi QUEUE

3.1 Operation Queues

- **Operation Object**
  - Một Operation objects là một thể hiện của lớp NSOperation được sử dụng để gói gọn công việc mà bạn muốn ứng dụng thực hiện.
  - Lớp NSOperation là một abstract class base(căn bản) bạn phải định nghĩa một subclass để có thể sử dụng hoặc sử dụng subclass có sẵn là NSInvocationOperation và NSBlockOperation.

3.1 Operation Queues

- **Một số Task của NSOperation:**
  - **Khởi tạo:**
    - **- init**: Trả về một đối tượng NSOperation được khởi tạo. Phương thức này chỉ sử dụng ở phiên bản iOS 2.0 đến iOS 7.1.
  - **Thực thi Operation:**
    - **- start**: Bắt đầu thực thi operation.
    - **- main**: Thực hiện nhiệm vụ không đồng thời. Mặc định khi triển khai phương thức này là một phương thức rỗng. Muốn thực hiện các nhiệm vụ cần thiết cần override phương thức này và định nghĩa các công việc cần thực hiện.
    - **completionBlock**: Trả về một khối (block) khi nhiệm vụ chính hoàn thành công việc. Block này không có tham số và không có giá trị trả về. Được gọi khi isFinished là YES.

3.1 Operation Queues

- **Một số phương thức và thuộc tính khác:**
  - **- cancel**: Dừng nhiệm vụ của một Operation
  - **queuePriority**: Trả về chỉ số ưu tiên của các operation trong hàng đợi.
  - **threadPriority**: Trả về mức ưu tiên của Thread đang hoạt động khi sử dụng operation. Mặc định là 0.5. Giá trị từ 0.0 – 1.0. Thuộc tính này chỉ có thể sử dụng ở iOS 4.0 đến iOS 8.0. Từ phiên bản 8.0 đã ngừng sử dụng.
  - - **waitUntilFinished**: Thực hiện khóa các thread hiện tại cho tới khi kết thúc.

3.1 Operation Queues

- **Giới thiệu về NSOperationQueue**
  - Một hàng đợi NSOperationQueue thực hiện hành động của một tập hợp các đối tượng NSOperation được định nghĩa.
  - Sau khi các đối tượng NSOperation được thêm vào NSOperationQueue một NSOperation sẽ hoạt động cho tới khi nó kết thúc và bị hủy.
  - Các NSOperation khác sau khi được thêm vào NSOperationQueue sẽ được tổ chức theo cấp độ ưu tiên và tùy vào độ ưu tiên sẽ được tiến hành tiếp theo.

3.1 Operation Queues

- **Một số Task của NSOperationQueue:**
  - **Quản lý Operation trong hàng đợi** :
    - - **addOperation**: Thêm một operation vào hàng đợi.
    - **- addOperations:waitUntilFinished**: Thêm một mảng operation vào hàng đợi
    - - **addOperationWithBlock**: Thêm một khối block để thực hiện operation kết quả trả về trong block sau khi đã thực hiện xong.
    - **operations**: Trả về một mảng mới chưa danh sách cách operation đang có trong hàng đợi theo thứ tự được thêm vào.
    - **operationCount**: Trả về số lượng operation hiện tại có trong hàng đợi.
    - - **cancelAllOperations**: Hủy bỏ tất cả các hàng đợi đang có và tất cả các operation có trong hàng đợi.
    - - **waitUntilAllOperationsAreFinished**: Khóa các thread hiện hành cho tới khi các operation và hàng đợi thực hiện xong.

3.1 Operation Queues

- **Một số phương thức và thuộc tính khác**:
  - **maxConcurrentOperationCount**: Trả về số lượng tối đa các operation hoạt động đồng thời cũng một lúc.
  - **suspend**: cho biết hàng đợi có đang được lên lịch để hoạt động không. Phương pháp này dùng để đình chỉ hoặc tiếp tục một hàng đợi.
  - **name**: Thuộc tính tên của hàng đợi.
  - **currentQueue**: Trả về hàng đợi đang hoạt động hiên tại. Là nil nếu không thể xác định.
  - **mainQueue**: Trả về hàng đợi hoạt động trên main thread.

3.1 Operation Queues

- **Thủ thuật khi sử dụng đối tượng Operation**
  - **Tránh bị Per-Thread Storage**
  - **Giữ tham chiếu đến các đối tượng Operation**
  - **Xử lý lỗi và ngoại lệ**
    - Quản lý bộ nhớ trong Operation Objects
    - Kiểm tra kiểu mã lỗi.
    - Kiểm tra lỗi phát sinh từ hàm nào.
    - Bắt ngoại lệ trong code của bạn hoặc farmework khác.
    - Bắt ngoại lệ NSOperation.
  - Lưu ý: Không được sửa đối tượng operation khi đã thêm vào hàng đợi.

3.2 Dispatch Queues

- **Giới thiệu về Dispatch Queues**
  - Grand Central Dispatch (GCD), dispatch queues là công cụ mạnh mẽ để thực hiện công việc hiệu quả.
  - Với dispatch queues bạn có thể giải quyết tất cả các công việc thực hiện trên các tiến trình khác nhau.
  - Ưu điểm của dispatch queues là rất dễ sử dụng, hiệu quả và đơn giản.

3.2 Dispatch Queues

- **Giới thiệu về Dispatch Queues**
  - Dispatch queues là một cách dễ dàng thể thực hiện công việc không đồng bộ (asynchronously) và đồng thời (concurrently) trong ứng dụng của bạn.
  - Ví dụ bạn có thể tạo ra các công việc như: tính toán, đọc ghi dữ liệu tập tin, truyền tải dữ liệu,.. Sau đó định nghĩa các công việc đó trong các hàm và thêm nó vào dispatch queues.

3.2 Dispatch Queues

- Một số Dispatch Queues có sẵn và cách dùng

Loại	Mô tả
Serial	Hàng đợi nối tiếp (còn gọi là private dispatch queues) thực hiện công việc trong một thời gian và thứ tự mà chúng đợi thêm vào hàng đợi. Mỗi công việc được chạy trên một thread riêng biệt. Thường được sử dụng để đồng bộ hóa truy cập vào resource. Bạn có thể tạo ra nhiều serial queue và chạy cùng lúc.
Concurrent	Concurrent queues (còn được gọi là global dispatch queue) thực hiện một hoặc nhiều công việc tại cùng một thời điểm và bắt đầu theo đúng thứ tự khi thêm vào hàng đợi. Công việc đang thực hiện chạy trên các thread khác nhau được quản lý bởi dispatch queue. Số công việc làm cùng lúc tại một thời điểm có thể thay đổi phụ thuộc vào điều kiện của hệ thống.
Main dispatch queue	Main dispatch queue là một globally available serial queue có sẵn thực hiện trên main thread của ứng dụng.

3.2 Dispatch Queues

- Ngoài dispatch queues, Grand Central Dispatch cung cấp một số công nghệ để có thể quản lý code của bạn.

Công nghệ	Mô tả
Dispatch groups	Một dispatch group là cách để quản lý tập hợp một block (bạn có thể theo dõi block synchronously hoặc asynchronously tùy thuộc vào nhu cầu của bạn).
Dispatch semaphores	Dispatch semaphores tương tự như semaphores nhưng hoạt động hiệu quả hơn
Dispatch sources	Dispatch sources tạo ra các thông báo để đáp ứng các loại sự kiện đặc biệt của hệ thống. Bạn có thể sử dụng Dispatch sources để theo dõi các sự kiện như : xử lý, thông báo, tín hiệu,.. Khi có một sự kiện dispatch source sẽ gửi mã công việc vào hàng đợi không đồng bộ specified dispatch queue để xử lý.

3.2 Dispatch Queues

- **Ví dụ:**

```
// tạo một dispatch queue
dispatch_queue_t myQueue = dispatch_queue_create("My Queue",NULL);

dispatch_async(myQueue, ^{
  // Thực thi công việc
  dispatch_async(dispatch_get_main_queue(), ^{
    // cập nhật UI
  });
});
```

- Tasks are represented as blocks of code.

- Only 1 task to be executed at a time.
- Serialized access to shared resource.

- No guarantee how many tasks are being executing at a time.
- No guarantee about execution time.
- No guarantee about order of execution.

System Queues:

Only **one** queue which is the **main queue**

```
let mainQueue = dispatch_get_main_queue()
```

System Queues:

4 queues called **global dispatch** queues.

- DISPATCH_QUEUE_PRIORITY_HIGH
- DISPATCH_QUEUE_PRIORITY_DEFAULT
- DISPATCH_QUEUE_PRIORITY_LOW
- DISPATCH_QUEUE_PRIORITY_BACKGROUND

They only differ in priority. **HIGH** is the highest priority and **BACKGROUND** is the lowest

```
let queue = dispatch_get_global_queue
    (DISPATCH_QUEUE_PRIORITY_HIGH, 0)
```

Creating your own queues

You can create any number of serial queues. The **first** param is a unique label to your queue

Creating your own queues

You can create any number of concurrent queues but you don't have to as you already have **4 global queues** to use.

3.3 Dispatch Source

Giới thiệu về Dispatch Source

- Dispatch source là một loại dữ liệu cơ bản điều phối việc xử lý của các sự kiện của hệ thống. Grand Central Dispatch hỗ trợ các loại dispatch sources như sau :
  - Timer dispatch sources : tạo ra các notification định kì.
  - Signal dispatch sources : thông báo tín hiệu khi có UNIX đến.
  - Descriptor sources: thông báo về các hoạt động của socket cơ bản của hệ thống.
  - Process dispatch source : thông báo cho bạn các sự kiện liên quan đến quá trình xử lý.
  - Mach port dispatch sources: thông báo cho bạn sự kiện liên quan tới mach.
  - Custom dispatch sources: Cho bạn khả năng tùy chỉnh.

3.3 Dispatch Source

❑ Bảng các hàm có thể gọi trong Event

Hàm	Mô tả
dispatch_source_get_handle	Hàm này trả về kiểu dữ liệu hệ thống cơ bản là dispatch source • Đối với descriptor dispatch source: trả về kiểu int chứa các mô tả kết hợp với dispatch source. • Đối với signal dispatch source: trả về kiểu int quy định tín hiệu số cho sự kiện gần nhất. • Đối với process dispatch source: trả về một cấu trúc dữ liệu cho quá trình theo dõi. • Đối với Mach port dispatch source: trả về một cấu trúc dữ liệu mach_port_t. • Đối với other dispatch sources: Không có giá trị trả về cho hàm này.
dispatch_source_get_data	Hàm này trả về dữ liệu đang chờ sự kết hợp với sự kiện.
dispatch_source_get_mask	Trả về flag cho sự kiện được dùng để khởi tạo dispatch source.

3.3 Dispatch Queues

- **Ví dụ:**

```
dispatch_source_t source =
dispatch_source_create(DISPATCH_SOURCE_TYPE_READ, myDescriptor,
0, myQueue);
dispatch_source_set_event_handler(source, ^{
  // Get some data from the source variable, which is captured
  // from the parent context.
  size_t estimated = dispatch_source_get_data(source);
  // Continue reading the descriptor...
});
dispatch_resume(source);
```

Thảo luận