Pace University - PCBT

Advanced iOS Class Spring 2015

Concurrency

- This lecture will provide a background and context for using concurrency and blocks within your iOS Apps.
- We will discuss the best practices in doing so.
- This lecture will also discuss many key topics such as NSOperation Queue .
- The lecture should be accompanied by an XCode exercise creating several projects implementing the topics.

What are Closures?

- Closures are self-contained blocks of functionality that can be passed around and used in your code. Closures in Swift are similar to blocks in C and Objective-C and to lambdas in other programming languages.
- Closures can capture and store references to any constants and variables from the context in which they are defined. This is known as closing over those constants and variables, hence the name “closures”. Swift handles all of the memory management of capturing for you.

What is Concurrency?

- In order to take advantage of multiple cores, a computer needs software that can do multiple things simultaneously. For a modern, multitasking operating system like OS X or iOS, there can be a hundred or more programs running at any given time, so scheduling each program on a different core should be possible. However, most of these programs are either system daemons or background applications that consume very little real processing time. Instead, what is really needed is a way for individual applications to make use of the extra cores more effectively.

Why not Threads?

- Although threads have been around for many years and continue to have their uses, they do not solve the general problem of executing multiple tasks in a scalable way. With threads, the burden of creating a scalable solution rests squarely on the shoulders of you, the developer. You have to decide how many threads to create and adjust that number dynamically as system conditions change. Another problem is that your application assumes most of the costs associated with creating and maintaining any threads it uses.

Grand Central Dispatch

- One of the technologies for starting tasks asynchronously is Grand Central Dispatch (GCD). This technology takes the thread management code you would normally write in your own applications and moves that code down to the system level. All you have to do is define the tasks you want to execute and add them to an appropriate dispatch queue. GCD takes care of creating the needed threads and of scheduling your tasks to run on those threads. Because the thread management is now part of the system, GCD provides a holistic approach to task management and execution, providing better efficiency than traditional threads.

Global and MainQueue

+ `dispatch_get_global_queue`
+ `dispatch_get_main_queue`

dispatch_async

```
let priority = DISPATCH_QUEUE_PRIORITY_DEFAULT
dispatch_async(dispatch_get_global_queue(priority,
0)) {

    // do some task that takes a long time
    dispatch_async(dispatch_get_main_queue()) {

        // update some UI
    }
}
```

dispatch_async

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QU
EUE_PRIORITY_BACKGROUND, 0), {

    println("This is run on the background queue")

    dispatch_async(dispatch_get_main_queue(), 0), {
        println("This is run on the main queue,
after the previous block")
    })

})
```

Communication with NSURLSession

```
let url = NSURL(string: "http://www.stackoverflow.com")

let task = NSURLSession.sharedSession().dataTaskWithURL(url)
{(data, response, error) in

    println(NSString(data: data, encoding:
NSUTF8StringEncoding))

}

task.resume()
```

What is this?

Closures look like:

```
{ (parameters) -> (return type) in

    (statements)

}
```

End of Lecture

rpascazio@pace.edu