CSC 491 / 391 Mobile Application Development for iOS II

Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu
@DePaulSWEng

1

Outline

- Multi-threads
- Grand Central Dispatch

2

Multi-Threads

3

Multi-Core Hardware

- iOS devices with multi-core CPU's
  - Recent models
  - 4-6 cores
  - 2 high-performance, 2-4 energy-efficiency
- Concurrent execution of code is possible
- Your code need to take advantage of the available hardware

4

Threads

- Lightweight mechanism to support multiple *concurrent* execution paths of code within an app
  - iOS supports multi-threading
  - Thread execution is managed by the system
- Potential benefits
  - Improve perceived responsiveness
    - Perform time-consuming tasks in the background
  - Improve real-time performance on multicore hardware

5

Manage Multiple Threads

- Each app has at least one thread – the *main* thread
  - The main thread handles all UI related tasks
  - Additional threads can be created if needed
- Complications of multiple threads within an app
  - Shared memory space.
  - All threads have access to all the data in the app
  - Uncoordinated access of data may lead to **data corruption, erroneous results,** and **unpredictable behaviors**
  - Subtle bugs, *Heisenbugs,* – dead lock, race condition, etc.

6

Thread Safety

- Thread safety
  - Measures to ensure proper behavior and results when data are shared by multiple threads
  - Carry performance overhead
- Most *Foundation* framework classes are *thread safe*
  - Safe to use in multi-threaded apps
- **UIKit is NOT thread safe**
  - Lightweight and responsive
  - All code accessing UI objects must run on the *main thread*
  - All UI-related tasks are executed on the same thread

DEPAUL UNIVERSITY 7

7

Approaches to Using Threads

- Use threads, `Thread` objects, directly
  - Low level, complicated, needs great care.

Not recommended

- *Grand Central Dispatch (GCD).*
  - An API wrapper for using threads
  - Hides some of the details of handling threads.
  - Easier to use. Less prone to errors
- *Timers*, `Timer` objects.
  - Simple mechanism to schedule tasks at regular intervals on the main thread.

DEPAUL UNIVERSITY 8

8

Grand Central Dispatch (GCD)

9

Grand Central Dispatch (GCD)

- A library to support multi-threading and concurrent execution of code
- System level support to accommodate all running apps
- Supports *synchronous* and *asynchronous* background tasks
- A simpler concurrency model
  - Defines *queues* and *tasks*
  - Handles scheduling of the tasks to appropriate threads.
  - No need to directly deal with *threads* and *locks* etc.
  - Avoid tricky concurrency related bugs

DEPAUL UNIVERSITY 10

10

Dispatch Queues & Tasks

- A *task* is a unit of work to be performed
  - Represented as a closure, i.e., a block of code
- *Dispatch queues* manage *tasks* to be executed
  - Queues are First-In-First-Out (FIFO)
  - Each queue has an associated thread to dispatch the tasks
- *Serial queues*

E.g., the main queue

  - Execute the tasks *sequentially*, one at a time, according to the order in the queue
- *Concurrent queues*
  - Execute the tasks *concurrently*, only ensure the starting order

DEPAUL UNIVERSITY 11

11

The Main Queue

- A special queue, associated with the *main thread*
  - A serial queue
- All UI related tasks **MUST** be performed in this queue and this queue only.
  - UIKit is not thread safe. Not safe for multi-threading
- And, conversely, only UI related tasks should occur on this queue.
  - All time-consuming or non-UI related tasks should **NOT** be performed in this queue.
  - To ensure that the UI is responsive!

DEPAUL UNIVERSITY 12

12

The Main Dispatch Queues

- The main queue

```
let mainQueue = DispatchQueue.main
```

- Dispatch a task *asynchronously* on a different queue

```
queue.async {
    … code …
}
```

  - Enqueue a task for *asynchronous* execution
  - Returns immediately
    - Move work off the main thread
    - Deferred execution of tasks
    - Automatic concurrency

DEPAUL UNIVERSITY 13

13

Dispatch Synchronously

- Dispatch a task *synchronously* on a different queue

```
queue.sync {
    … code …
}
```

- Enqueue a task for *synchronous* execution
- Does not return until the block is complete
- Execute the block on the same thread when possible

DEPAUL UNIVERSITY 14

14

Perform UI Related Tasks Safely

- To off load some tasks to other queues
- Dispatch a task from the main queue to another

```
notTheMainQueue.async {
    Some time consuming tasks, no UI access
    DispatchQueue.main.async {
        Update UI with progress or results
    }
}
```

Update to UI must be executed on the main queue

DEPAUL UNIVERSITY 15

15

Other System Queue

- Several *concurrent* queues are provided by the system
- Target different levels of *quality of service* (QoS)
- Enum `DispatchQoS.QoSClass`

`.userIteractive`	*quick and high priority, the main thread*
`.userInitiated`	*high priority, immediate result*
`.utility`	*long running*
`.background`	*not visible to user (prefetching, etc.)*

- Get the system queue with a given *qos*

```
DispatchQueue.global(qos: qos )
```

DEPAUL UNIVERSITY 16

16

Create Serial Queues

- You can create your own *serial* queue if needed

```
let serialQueue = DispatchQueue(label: "name")
```

- Need the tasks to be executed sequentially
  - Downloading a bunch of things from a certain website but you don't want to deluge that website, so you queue the requests up serially
  - The tasks to be performed depend on each other in a serial fashion

DEPAUL UNIVERSITY 17

17

Dispatch Time

- Class `DispatchTime`
  - A point in time, with nanosecond precision
  - Relative to the system clock, which may sleep
  - Type method: `now()`
  - Constant: `forever`
- Class `DispatchWallTime`
  - An absolute point in time, with microsecond precision
  - Based on the "Wall Clock"
  - Type method: `now()`
  - Constant: `forever`

DEPAUL UNIVERSITY 18

18

Dispatch Time Interval

- Enum `DispatchTimeInterval`
  - Number of seconds, millisconds, microseconds, or nanoseconds.
  - Cases:
    - `.seconds(Int)`
    - `.milliseconds(Int)`
    - `.microseconds(Int)`
    - `.nanoseconds(Int)`

Enum with associated values

DePaul University 19

19

Dispatch Time Interval

- Enum with associated values

```
enum DispatchTimeInterval {
    case seconds(Int)
    case milliseconds(Int)
    case microseconds(Int)
    case nanoseconds(Int)
}
```

- Operators: where *T* can be *Dispatch Time* or *Dispatch Wall Time*
  - *T* + *DispatchTimeInterval* *T* + *Int*
  - *T* – *DispatchTimeInterval* *T* – *Int*

In seconds

DePaul University 20

20

Dispatch with a Delay

- Use the `asyncAfter` method of a dispatch queue
  - Specify a deadline
- Dispatch time can be specified in the form of
  - time + interval

```
let time = DispatchTime.now() + .seconds(3)
DispatchQueue.main.asyncAfter(deadline: time) {
    Do something in the future on the main queue
}
```

DePaul University 21

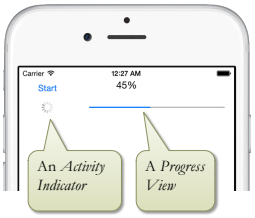
21

Demo
Progress View

22

The Progress View App

- Simulating a time consuming task triggered by the "*Start*" button.
- An animated *Activity Indicator,* while the task is in progress
  - Select "*Hidden when stopped*"
- A *Progress View* showing the progress of the task



DePaul University 23

23

The Outlets

- Connect outlets to the *Activity Indicator*, the *Progress View* and the *Label* in the view

```
class ViewController: UIViewController {

    @IBOutlet weak var indicator: UIActivityIndicatorView!

    @IBOutlet weak var progress: UIProgressView!

    @IBOutlet weak var label: UILabel!

    …
}
```

DePaul University 24

24

The Action

```
@IBAction func start(_ sender: UIButton) {
    indicator.startAnimating()
    let queue = DispatchQueue.global(qos: .background)
    queue.async {
        for i in 0 ... 100 {
            DispatchQueue.main.async {
                self.progress.progress = Float(i) / 100
                self.label.text = "\(i)%"
                if (i == 100) {
                    self.indicator.stopAnimating()
                }
            }
            usleep(100_000) // in microseconds
        }
    }
}
```

The system queue for background tasks

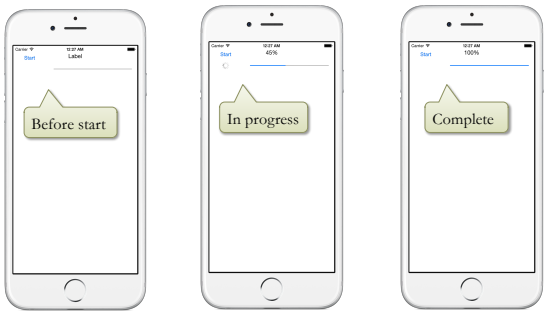
Update UI

Sleep for 0.1s. Do work in the background

DePaul University 25

25

Run the Progress View Demo



DePaul University 26

26

Next …

- Error handling
- Background processing

✧ Xcode, iOS, WatchOS are trademarks of Apple Inc.

DePaul University 27

27