Concurrency in iOS

Jeff Kelley | @SlaunchaMan
www.jeffkelley.org
Strange Loop | September 24, 2012

Who?

DETROIT
LABS

Concurrency

- It isn't enough to go *fast*
  - Moore's Law expiring early
- Expanding to multiple processor cores, not faster processors
- Manually creating threaded code sucks
- Different tools for different jobs

Going Fast

- Processors are still getting faster, but it’s slowing down
- This was predicted for 2015, but something funny happened along the way
- Mobile processors have more stringent heat and power consumption needs

Going Fast

- Desktop computers are going multicore
- A Mac Pro can have twelve processor cores!
- The fastest possible algorithm may not matter if it uses a single core

Threaded Code

Some people, when confronted with a problem, think, "I know, I'll use threads," and then two they hav erpoblesms.

Ned Batchelder

Threaded Code

- Manually-threaded code is horrible to write
  - Query the number of cores
  - Ask them how busy they are
  - Create the appropriate number of threads
  - Do stuff on those threads, monitoring the cores to see which one to use

Yuck.

UNIX Threading

UNIX Threading

- Full support for the things you already know from UNIX and BSD
  - pthreads, kqueues, etc.
- Extremely low-level, but powerful

NSThread

- Objective-C threading API
- Higher-level than UNIX threads, still expose raw details
- Still have to manually create/destroy threads

Threading Problems

- It’s difficult to gauge current CPU use and impossible to know future use
- Two programs each trying to be as multithreaded as possible will fight for resources
- Lots of wasted effort and surface area for bugs
- Bugs here are harder to track down and potentially extremely nasty

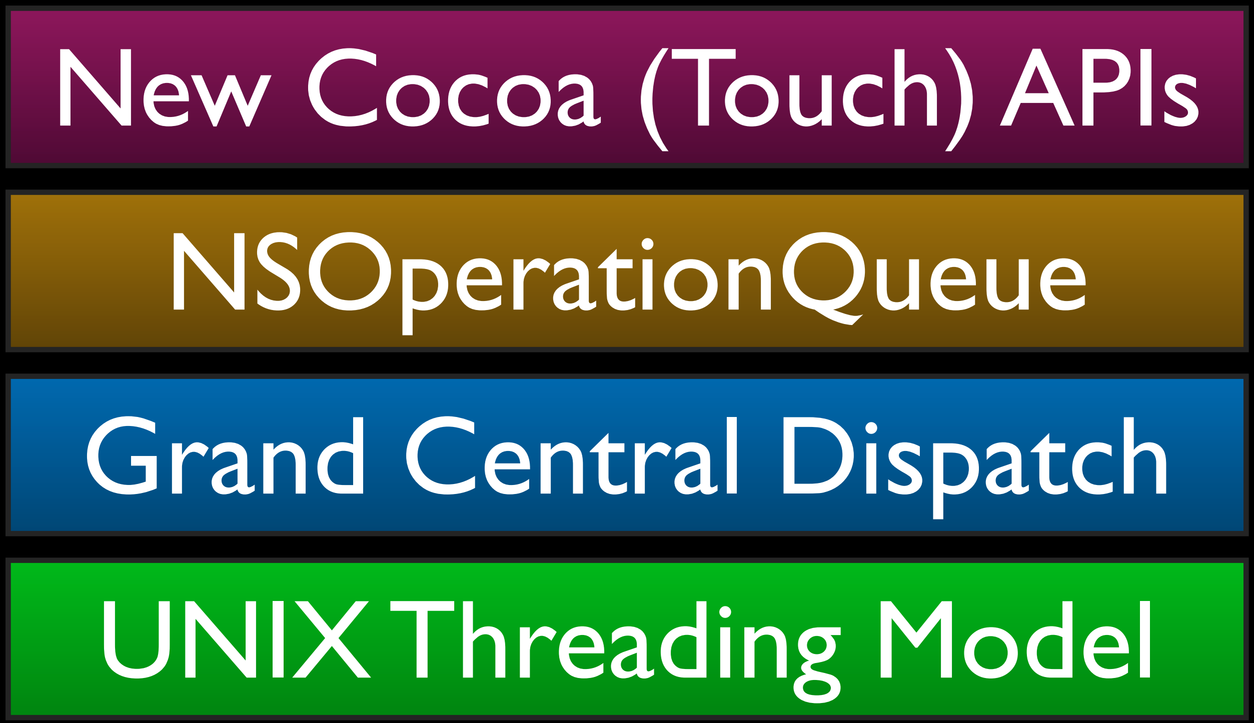
Thread Safety

- Writing to a portion of memory on one thread while trying to read that portion of memory on another is… problematic.
- All kinds of solutions for this
  - @synchronize(myObject)
  - Locks, semaphores, etc.
  - Core Data “thread safety”

Thread Safety

- This is one problem we won’t solve today.
- We *will* make it better.

So What's a Developer To Do?



New Cocoa (Touch) APIs

NSOperationQueue

Grand Central Dispatch

UNIX Threading Model

So What's a Developer To Do?

- Stop managing threads on your own
- Think of the things your app needs to do as *units of work*.
- Enqueue units of work and let the OS decide how to run them
  - The OS has a lot more knowledge than your program does

Grand Central Dispatch

Grand Central Dispatch

- C API for managing queues of work
- Relies heavily on blocks, an Apple extension to the C language
- Manually memory managed
- Open-sourced as *libdispatch*
- Generally pretty awesome

Simple GCD

```
dispatch_queue_t queue =
dispatch_get_global_queue(DISPATCH_QUEUE_
PRIORITY_HIGH, 0);

dispatch_async(queue, ^{

   [self performLongTask];

});
```

Simple GCD

```
dispatch_queue_t queue =
dispatch_get_global_queue(DISPATCH_QUEUE_
PRIORITY_HIGH, 0);

dispatch_async(queue, ^{

   [self performLongTask];

});
```

Simple GCD

```
dispatch_queue_t queue =
dispatch_get_global_queue(DISPATCH_QUEUE_
PRIORITY_HIGH, 0);

dispatch_async(queue, ^{

   [self performLongTask];

});
```

Simple GCD

```
dispatch_queue_t queue =
dispatch_get_global_queue(DISPATCH_QUEUE_
PRIORITY_HIGH, 0);

dispatch_async(queue, ^{

   [self performLongTask];

});
```

Simple GCD

```
dispatch_queue_t queue =
dispatch_get_global_queue(DISPATCH_QUEUE_
PRIORITY_HIGH, 0);

dispatch_async(queue, ^{

   [self performLongTask];

});
```

Simple GCD

```
dispatch_queue_t queue =
dispatch_get_global_queue(DISPATCH_QUEUE_
PRIORITY_HIGH, 0);

dispatch_async(queue, ^{

   [self performLongTask];

});
```

Basic Dispatch Functions

- dispatch_async(queue, block);
  dispatch_async_f(queue, context, func);
  - Schedules block or function on queue, returns immediately
- dispatch_sync(queue, block);
  dispatch_sync_f(queue, context, func);
  - Schedules block or function on queue, blocks until completion

Dispatch Queues

- dispatch_queue_t
- Main Queue
  - Analogous to main thread (Do your UI operations here)
  - dispatch_get_main_queue()

Global Queues

- dispatch_get_global_queue(priority, flags);
  - priority is one of four constants:
    - DISPATCH_QUEUE_PRIORITY_BACKGROUND
    - DISPATCH_QUEUE_PRIORITY_LOW
    - DISPATCH_QUEUE_PRIORITY_NORMAL
    - DISPATCH_QUEUE_PRIORITY_HIGH
  - flags arg should always be 0 (for now)

Making Queues

- dispatch_queue_create(label, attr)
  - Use reverse DNS for label
    - com.example.myQueue
  - attr defines the type of queue
    - DISPATCH_QUEUE_SERIAL
    - DISPATCH_QUEUE_CONCURRENT
- Be sure to use dispatch_release()

Using Queues

- The main queue is serial
  - First-in, first-out, one at a time
- Global queues are concurrent
  - GCD automatically chooses how many (usually # of CPU cores)
- You pick for queues you create

Queues To Control Access

- Easy way to limit access to a piece of memory
    - Create a serial queue (one-at-a-time, FIFO) for the object
    - All access to the object goes through this queue
    - No lock required!

Typical GCD Pattern

- dispatch_async() with a background queue to kick off work
- dispatch_async() with the main queue to display the results

Demo

Grand Central Dispatch

- Useful for more than just threading!
  - Can be used to replace the main run loop in your app
    - For a good, lightweight example of a C program using GCD, check out the source to Mountain Lion’s *caffeinate* utility
- Can support timers and file notifications

Grand Central Dispatch

- Manages threads for you, uses as many as it needs
- Not the most user-friendly API in the world
  - No way to cancel a task
  - No way to adjust the priority of a task
  - Memory Management?!?

NSOperationQueue

NSOperationQueue

- Much like GCD, you enqueue units of work onto queues
- Unlike GCD, the units of work and the queues themselves are Objective-C objects
  - NSOperation and NSOperationQueue

NSOperationQueue

- Operations can have priority amongst one another
  - `[myOperation setQueuePriority:NSOperationQueuePriorityLow];`
- Operations can depend on one another
  - `[myOperation addDependency:myOtherOperation];`
  - Even across different queues!

NSOperationQueue

- Operations are cancellable
  - [myOperation cancel];
- In your custom operation class, check for the canceled property

Custom Operation Class?

- Two ways to create an operation
  - NSBlockOperation
    - Create an operation with a work block
  - Subclass NSOperation
    - Implement -main with your custom logic

Why Subclass

- Gives you a pointer to self to call [self isCancelled]
- Asynchronous operations
  - URL loading, geocoding, etc.
  - The end of main does not necessarily end the operation
  - Implement -start and -isFinished

Demo

NSOperationQueue

- Objective-C class to manage the execution of units of work
- Create custom operations to perform a unit of work
- With ARC, you don’t need to worry about memory management
- Can cancel and prioritize tasks

New Cocoa (Touch) APIs

New Cocoa (Touch) APIs

- Sometimes you don’t want to worry about managing threads, dispatch queues, or operation queues
- Common, repetitive tasks that could be made faster with concurrency, but it’s not worth the effort to create a queue and manage it
- Apple wants you to write fast code

New Cocoa (Touch) APIs

- Enumerating a Collection
- Sorting an Array

Enumerating a Collection

- A task as old as programming itself
- Walk the collection, item-by-item, and do something with each one

Enumerating a Collection

```
NSUInteger count = [myArray count];
for (int i = 0; i < count; i++) {
    id obj = [myArray objectAtIndex:i];
    [obj doSomething];
 }
```

Enumerating a Collection

```
NSUInteger count = [myArray count];
for (int i = 0; i < count; i++) {
    id obj = [myArray objectAtIndex:i];
    [obj doSomething];
    for (j = 0; j < [myNewArray count]; j++) {
      // More code inside this loop!
    }
 }
```

Enumerating a Collection

```
NSEnumerator *enum = [myArray objectEnumerator];
id object;
while ((object = [enum nextObject])) {
   [object doSomething];
}
```

Enumerating a Collection

```
NSEnumerator *enum = [myArray objectEnumerator];
id object;
while ((object = [enum nextObject])) {
   [object doSomething];
   NSUInteger i = [myArray indexOfObject:object];
}
```

Enumerating a Collection

```
for (id object in myArray) {
   [object doSomething];
}
```

Enumerating a Collection

```
size_t count = [myArray count];
dispatch_queue_t queue = ...
dispatch_apply(count, queue, ^(size_t i) {
   id object = [myArray objectAtIndex:i];
   [object doSomething];
});
```

Enumerating a Collection

```
[myArray
enumerateObjectsWithOptions:NSEnumerationConcurrent
usingBlock:^(id obj, NSUInteger idx, BOOL *stop) {

  [obj doSomething];

}];
```

Enumerating a Collection

```
[myArray
enumerateObjectsWithOptions:NSEnumerationConcurrent
usingBlock:^(id obj, NSUInteger idx, BOOL *stop) {

   [obj doSomething];

}];
```

Enumerating a Collection

```
[myArray
enumerateObjectsWithOptions:NSEnumerationConcurrent
usingBlock:^(id obj, NSUInteger idx, BOOL *stop) {

  [obj doSomething];

}];
```

Enumerating a Collection

- Concurrency for free!
- Don’t worry about queue management
- Very quickly add concurrency to an existing project

Sorting a Collection

- NSArray and NSOrderedSet collections sometimes need sorting
- Many, many algorithms
- The more objects in the collection, the more time it's going to take—potentially exponentially

Sorting a Collection

```
NSMutableArray *myArray;

[myArray sortWithOptions:NSSortConcurrent
         usingComparator:^NSComparisonResult(id obj1, id obj2) {
           return [obj1 compare:obj2];
       }];
```

Sorting a Collection

```
NSMutableArray *myArray;

[myArray sortWithOptions:NSSortConcurrent
         usingComparator:^NSComparisonResult(id obj1, id obj2) {
           return [obj1 compare:obj2];
       }];
```

Sorting a Collection

```
NSMutableArray *myArray;

[myArray sortWithOptions:NSSortConcurrent
         usingComparator:^NSComparisonResult(id obj1, id obj2) {
           return [obj1 compare:obj2];
       }];
```

Sorting a Collection

```
NSMutableArray *myArray;

[myArray sortWithOptions:NSSortConcurrent
         usingComparator:^NSComparisonResult(id obj1, id obj2) {
           return [obj1 compare:obj2];
       }];
```

Sorting a Collection

- Stop worrying about sort algorithm (for most applications)
- Utilize as many cores as needed to sort your data
- Huge returns as hardware increases in throughput

Thread Safety

- Don’t modify objects from multiple queues
- Use dispatch queues to coordinate access
- Use the main dispatch and operation queues for UIKit operations
- Assume Apple code is *not* thread-safe

GCD Barriers

- Great tool for thread safety
- Allow for concurrent reading of data but serial writing
- For instance, read from a dictionary on any queue simultaneously, write to it on a single queue

Demo

Thread Safety and Core Data

- Create a separate Managed Object Context for each queue
- Don't pass NSManagedObject instances between queues
  - Use the object ID instead
- Register for the NSManagedObjectContextDidSaveNotification notification

Wrap-Up

- Concurrency is an enormous topic
- Thread Safety is its own talk, especially if you use Core Data
- Concurrency is not magic performance snake oil
- Concurrency *does* help you take advantage of hardware enhancements

For More Info

- http://jeffkelley.org
- @SlaunchaMan
- github.com/SlaunchaMan
- *Learn Cocoa Touch*