University of Regina

CS 455/855
Mobile Computing

Advanced Networking

Dr. Orland Hoeber
orland.hoeber@uregina.ca
http://www.cs.uregina.ca/~hoeber/455/2018F

Readings

- Online documentation for
  - Grand Central Dispatch
  - MultiPeer Connectivity
  - Push Notifications

Advanced Networking Topics

- Grand Central Dispatch
  - mechanism for building multi-threaded execution queues
- Multipeer Connectivity
  - method for creating client-server or peer-to-peer sessions on a local network
- Push Notifications
  - method for allowing external events to initiate communication to your device

Grand Central Dispatch

- Multi-threaded programming can be complex, since most languages and OSs require that you manage the forking and joining of threads manually

- In the philosophy of providing high-level APIs to hide the complexity of low-level programming, iOS provides Grand Central Dispatch (GCD)
  - API for concurrency programming
  - uses dispatch queues
  - the OS takes care of thread management for you
  - the OS can optimize the number of threads based on the number of cores in the CPU

Dispatch Queues

- The core of GCD that allows for parallelism and concurrent execution are the dispatch queues
  - serial queues
    - guarantee that only one task runs at a given time
    - execution is done in-order (fifo)
  - concurrent queues
    - multiple items may be run at the same time
    - tasks are guaranteed to start in the order they were added (but there are no guarantees on the order of completion)
- The decision about when to start a task and which threads they run on is entirely handled by GCD

GCD Queue Types

- There are three main types of dispatch queues provided by GCD:
  - main queue
    - serial queue
    - always runs on the main thread
  - global queues
    - concurrent shared queues, differing in their priority of execution
      - high
      - default
      - low
      - background
  - custom queues
    - your own queues, which eventually are handled by one of the above global queues

QoS Property

- When you access the queues, you don't specify the priority directly
- Instead, you specify the Quality of Service that is needed, and let GCD decide what to do
  - user-interactive
    - interface updates that require low latency
    - mapped to the main queue
  - user-initiated
    - user tasks are critical, so this is mapped to the high priority global queue
  - utility
    - potentially long running tasks, computations, networking, etc.
    - mapped to the low priority global queue
  - background
    - tasks that the user isn't aware of happening (e.g., pre-fetching data)
    - mapped to the background priority global queue

Synchronous vs. Asynchronous

- When accessing a GCD queue, you need to decide whether the task should be executed synchronously or asynchronously
  - synchronous
    - function returns control to the caller after the task is complete
  - asynchronous
    - function returns immediately, allowing the code block to continue to execute (e.g., does not block the main thread)
- By default, you should use asynchronous queues since they promote the parallelism and concurrency that we are trying to achieve
- Some networking classes (URLSession) require the use of serial queues to ensure that callback functions are executed and completed in-order (when in doubt, use the default or check the documentation)

Example Code

- We have already used GCD in some of our network programming
  - the URLSessionDataDelegate methods are automatically executed within a GCD queue that is optimized for network communication
  - within those methods, if we want to update the interface, we need to push that code execution onto a user-interactive queue

```
DispatchQueue.main.async {
    self.rawText.text = dataString
}
```

Another Example

- You can also explicitly nest your dispatch queue access

```
DispatchQueue.global(qos: .userInitiated).async{
    // do some work here that was initiated by the user
   DispatchQueue.main.async{
      // use the above work to update the interface
   }
}
```

- Most of the time, you can just use the main and global queues

Recap

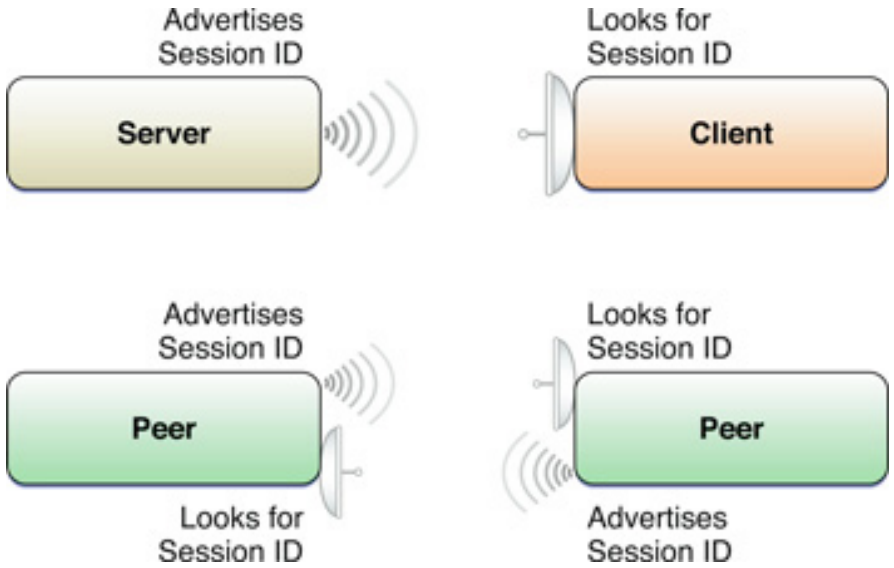
- GCD is the mechanism for configuring multi-threaded programming
- The value of GCD is that you don't have to worry about the threads themselves
- Some cautions:
  - be honest in your use of the QoS parameter (userInteractive, userInitiated, utility, background)
  - moving code off of the main thread will not solve the performance issues caused by inefficient algorithms
  - don't forget to move any code that needs access the UI elements back onto the main thread

Multipeer Connectivity

- The multipeer connectivity (MC) framework supports the discovery of services and communication with those services for nearby devices
  - creates an ad hoc network between multiple devices in the same local area
  - may be client-server or peer-to-peer
  - allows for communications between devices over networking infrastructure that happens to be available:
    - WiFi if the devices are on the same network
    - peer-to-peer WiFi if neither is connected to a WiFi network
    - Bluetooth if the devices are near enough to one another
  - the low-level networking is handled automatically based on which networking resources are available

Client vs. Server vs. Peer

- It is important to understand the distinction between clients, servers, and peers
  - clients – initiate communication
  - servers – respond to communication requests
  - peers – act as both client and server



Multipeer Connectivity Classes

- MCPeerID
  - display name to show to other devices
- MCNearbyServiceAdvertiser & MCNearbyServiceAdvertiserDelegate
  - tell other devices that you are available and manage the invitations made to communicate
- MCNearbyServiceBrowser & MCNearbyServiceBrowserDelegate
  - scan for advertised services and manage connections/disconnections
- MCSession & MCSessionDelegate
  - object that gets told to send the data
  - delegate methods to handle the receiving of the data

Multipeer Connectivity

- MCSession supports peer-to-peer communication within a multi-peer connectivity session
- What does this mean?
  - the peer group can contain more than two peers
  - communication can be between subsets of the group, or broadcast to everyone
- Use:
  - multi-player games
    - communication between teams
    - communication between all players
  - coordination between nearby devices

MCSession and Delegates

- Steps:
  - initialize the MCSession object
  - either advertise that you are willing to join the specified group, or invite peers to your session
  - use delegation to detect changes in state and communication
    - session(_:peer:didChange:)
    - send(_:toPeers:with:)
    - session(_:didReceive:fromPeer:)
  - all delegate methods operate on a private dispatch queue, similar to the URLSession object
  - this means that any interface changes need to be done on the main thread using GCD

Protocols for Communication

- MCSession provides the mechanism for connecting local devices and for sending and receiving data
- However, it does not enforce any details regarding the communication protocol
  - all that is sent is a Data object
  - you must decide
    - what to put in this object
    - whether to send it in a reliable or unreliable manner
    - whether to acknowledge received data
    - how to validate and process the data
    - what to do with the data within the context of the application

Recap

- Peer to Peer networking is supported by MCSession (MultipeerConnectivity)
- The methods hide the actual communication mechanism
- The coding is very similar to that of URLSession, but in the context of multipeer communications rather than client-server communications

Push Notifications

- There are often situations where some event occurs external to the device for which we wish to inform the user
  - examples: news/score alerts, availability/unavailability of some service, completion of some activity, etc.
  - push notifications is a mechanism for allowing these events to post messages to the user of the device
    - the user can choose to dismiss the message or open the associated application
    - if the associated application is opened, it can then access the full details associated with the specific notification

Push Notification Server

- Implementing push notifications requires the cooperation between a number of different parties
  - your app and your server
  - your server and Apple's push notification server

- This cooperation is enabled through with proper settings of the provisioning profile for your app
  - configured in the iOS Provisioning Portal
    - app id that does not contain wild-cards
    - push notification enabled
    - provisioning profile generated with this app id

Push Notification Server

- Four-step process to setup push notifications:
  - The UIApplication method registerUserNotificationSettings asks the user to confirm the types of notifications to accept
  - The UIApplication method registerForRemoteNotificatios() communicates with the push notification server (asynchronously) to generate a token identifying this instance of your app
  - this token is received by the UIApplicationDelegate method application(application: didRegisterForRemoteNotificationsWithDeviceToken:)
  - the token must then be sent back to your server (the server that will eventually wish to send push notifications to your app)
    - the server must store all of these tokens (one for each instance of the app) and use them when sending a notification

Sending a Notification

- Sending a notification consists of the following steps (on the server)
  - connect to Apple's push notification server
  - for each instance of the application that is to receive the notification (tokens received), a message is streamed to the push notification server
    - token
    - notification (JSON format)
      - message to show the user on their device (alert)
      - application-specific data

Receiving a Notification

- If push notifications are turned on on the device, it will open a persistent and secure connection to a push notification server
  - if there is a message, it will be delivered to the device
  - if the target app is not running, the user message will be shown to the user in a modal window
    - user can close the notification, or view details which will open the app and deliver the entire message to the app
    - delivered through application(application: didFinishLaunchingWithOptions:)
  - if the target app is running, the entire message will go directly to the app
    - through application(application: didReceiveRemoteNotification:)

UserNotifications Framework

- The push notifications are wrapped in a more general UserNotification framework
  - common method for push notifications (user notification), regardless of the source
  - Server: using Apple Push Notification Service (APNs)
  - Local:
    - Specific date/time
    - Time interval
    - Location
    - Build you own custom subclass of UNNotificationTrigger

Recap

- Sending a push notification from a remote server needs to have tokens for each instance of the app that is to receive the message
  - the logic for knowing who gets what message must be implemented on your server
- Messages are sent to Apple's push notification server
- Devices with push notification enabled will periodically poll the push notification server looking for messages
- Local notifications work in a very similar manner, but are initiated within the device rather than from some external source

Homework

- Next topic: Persistent Data Storage
- Project Milestone 2: Project Update
  - due Nov 16
- Short Paper #2 (CS 855)
  - due Nov 23