CSIT242 Mobile Application Development

LECTURE 7-1 – THREADS, PERFORMANCE, OPTIMIZATION, SECURITY

Outline

- Threads and multithreaded apps
- App performance & optimization
- Security


Threads

- Threads are a relatively lightweight way to implement multiple paths of execution inside of an application
- From a technical standpoint, a thread is a combination of the
  - kernel-level structures - that coordinate the dispatching of events to the thread and the preemptive scheduling of the thread on one of the available cores
  - application-level structures - include the call stack for storing function calls and the structures the application needs to manage and manipulate the thread's attributes and state
- In a non-concurrent application, there is only one thread of execution
  - This thread starts and ends with the application's main routine and branches one-by-one to different methods or functions
  - This thread must respond to events, update application's windows, and perform all of the computations needed to implement application's behavior
- Multithreaded application - An application that supports concurrency starts with one thread and adds more as needed to create additional execution paths
  - Each new path has its own custom start routine that runs independently of the code in the application's main routine

Multithreaded applications

- Multithreaded applications advantages:
  - Multiple threads can improve an application's perceived responsiveness
  - Multiple threads can improve an application's real-time performance on multicore systems
- Multithreaded applications challenges:
  - Each thread has to coordinate its actions with other threads to prevent it from corrupting the application's state information
  - If two threads try to manipulate the same data structure at the same time, one thread might overwrite another's changes in a way that corrupts the resulting data structure
- Possible solutions for multithreaded challenges:
  - Make sure each thread has its own distinct set of resources on which to operate
  - Synchronize access to the resource using locks, conditions, atomic operations, and other techniques

App’s performance and optimization

- The app will be running on multiple types of hardware, different processors running at different speeds, and available memory resources
- Choosing the right algorithms and data structures should always be priority There are two basic rules for writing efficient code:
  - Don't do work that you don't need to do
  - Don't allocate memory if you can avoid it
- The goal of optimizations should be to do the most work in the most efficient way possible
- You should always optimize your app’s algorithms using different tools

App's performance and optimization

- When using images:
  - Use *png* - *png*s are the best for performance as they are supported by the GPU
  - Use images that are the correct size for the cell; Resizing an image to fit can be expensive, especially if you're scrolling through hundreds of cells
  - Compress the images as much as possible

App's performance and optimization

- Static analysis (static code analysis) - analyses the code for inefficiencies and problems
  - The process provides an understanding of the code structure, and can help to ensure that the code adheres to industry standards
- Automated tools that can assist programmers and developers in carrying out static analysis
  - **The Static Analyzer (iOS)** - tries out thousands of possible code paths in a few seconds, reporting potential bugs that might have remained hidden or bugs that might be nearly impossible to replicate; This process also identifies areas in the code that don't follow recommended API usage, such as Foundation, UIKit, and AppKit idioms
  - **Android Lint** - a tool that scans Android project sources for potential bugs; Some examples of the types of errors that can looks for: missing and unused translations, layout performance problems, unused resources, accessibility and internationalization problems, icon problems, usability problems, manifest errors, etc

App's performance and optimization

- Power consumption
  - When using the hardware on any mobile device, you should always let the chip idle; This rule applies to the radios (Bluetooth, WiFi, GPS), hardware devices (Accelerometer, Gyroscope, Compass) and the CPU
  - If you need to use the network or the CPU, you should always execute your code in bursts; Once the code execution is complete, then the CPU/radios/other devices will be able to rest
  - You should not be polling for information; If you continuously poll the network/hardware for changes, then you are continuously using CPU cycles unnecessarily
  - Your best option is to use delegate methods that are provided, callback methods in blocks, listeners; Use frameworks, libraries and packages rather than creating your own classes
  - Move work off the main thread
  - Avoid blocking on threads
- This allows for much better battery life in your application

App’s performance and optimization

- Android Device Monitor
  - Android Device Monitor is a standalone tool that provides a UI for several Android app debugging and analysis tools. Most components of the Android Device Monitor are deprecated in favour of updated tools available in Android Studio 3.0 and higher
- Android Profiler
  - window in Android Studio 3.0 and higher that replaces the Android Monitor tools.
  - Provides real-time data for the app's CPU, memory, and network activity
  - Can perform sample-based method - trace the time of code execution, capture heap dumps, view memory allocations, and inspect the details of network-transmitted files
- Other useful tools for android apps:
  - Graphics API Debugger, Layout Inspector

App's performance and optimization

- Instruments is a separate app, which may be used independently as needed or can be embedded and used within Xcode to
  - trace different aspects of the apps, processes, and devices over time
  - collects data as it profiles, and presents the results in details for analysis
- Can be used to measure:
  - Memory Allocation, Memory Leaks
  - CPU usage
  - Energy Usage
  - System Usage
  - Track down problems in the source code

Security

- In the cloud-enabled, highly networked world of modern computing, security is one of the most important facets of proper software engineering
- It's important to ensure the security of the user's device and the applications running on it
- You must consciously design security into your app or service from the very beginning, and make it a conscious part of the entire process from design through implementation, testing, and release

Security

- At the application layer, security means being aware of how your code uses information and ensuring that it does so safely and responsibly
  - Keep users' personal data safe
    - Store the data in a secure way, and ensure that your software collects only the information that it requires
  - Protect data in transit
    - If your software transmits personal information over the Internet, you must do so in a safe and secure fashion to prevent unauthorized access to or modification of the data while in transit
  - Treat untrusted files and data with care
    - If your software accesses the Internet or reads files that might have previously been sent to someone over the Internet, your software must properly validate the data
  - Verify the authenticity of data where possible
    - If your software provides access to or works with signed data, it should verify those signatures to ensure that the data has not been tampered with

Security

- The Secure Enclave is a secure coprocessor that includes a hardware-based key manager
  - It is isolated from the main processor to provide an extra layer of security
  - It's also responsible for processing fingerprint and face data (from the Touch ID and Face ID sensors) for providing secure authentication while keeping user biometric data private and secure
- iOS employs many techniques to ensure device and application security and gives developers many easy ways to secure their data from malicious attacks
  - iOS prevents casual device access
    - the passcode allows data to remain encrypted until the user enters their passcode
  - When the device is locked, the user's data should be secure
    - The passcode protects the data in the keychain or on the file system
    - iOS encrypts the data using 'Data Protection' using the user passcode as the key to decrypt it
  - The file is protected by a randomly generated file key that is created
    - The contents of a file may be encrypted with one or more per-file (or per-extent) keys
    - These keys are wrapped with a class key and stored in a file's metadata
    - Files' metadata is encrypted with the file system key
    - The class key is protected with the hardware UID, and for some classes the user's passcode

Source: https://manuals.info.apple.com/MANUALS/1000/MA1902/en_US/apple-platform-security-guide.pdf

Security

- iOS restricts an application to a unique and secure location in the file system, also known as its “sandbox”
  - The sandbox limits the amount of access the application has to files, preferences, network resources and hardware
- Pre-iOS 6, users were only notified when applications were accessing their location From iOS 6 the system now asks for permission for:
  - Contacts, Calendars, Reminders, Photo Library
- Digital signatures are required on all applications for iOS
  - Code signing ensures the integrity of the program and allows the system to recognise updated versions as the same program as the original
  - Any change in code not intended by the developer can be detected by the system
  - Any application installed and not signed by Apple will not execute

Security

- Android is quite secure operating system designed to be truly open with a multi-layered security architecture
  - Linux Security - Linux-based kernel that is highly stable and secure kernel
    - Prevents user A from reading user B's files
    - Ensures that user A does not exhaust user B's memory
    - Ensures that user A does not exhaust user B's resources (CPU , GPS, Bluetooth, telephony)
  - Security-Enhanced Linux (SELinux) enforce mandatory access control over all processes - applying access control policies (from Android 4.3)
    - SELinux can be used to label these devices so the root can write to only those specified in the associated policy; Root cannot overwrite data and system settings
    - This prevents potentially compromised processes running as root from access symlinks, system files, app data and setattr

Security

- Kernel-level Application Sandbox
  - Assigns a unique user ID to each Android application and runs it as that user in a separate process
  - If application A tries to do something malicious like read application B's data, then the operating system protects against this because application A does not have the appropriate user privileges
- Runtime Permissions – give more granular controls to the user

Security

- Encryption is the process of encoding all user data on an Android device using symmetric encryption keys, all user-created data is automatically encrypted before committing it to disk and all reads automatically decrypt data before returning it to the calling process
  - Full-disk encryption
    - Android 5.0 to Android 9 supports full-disk encryption (uses a single key protected with the user's device password); Since access to data is protected behind their single user credential some features can be unavailable immediately after reboot
  - File-based encryption
    - Introduced in Android 7.0 allows different files to be encrypted with different keys that can be unlocked independently; Also, apps can operate within a limited context
  - Metadata encryption
    - With metadata encryption, a single key present at boot time encrypts whatever content is not encrypted by FBE, such as directory layouts, file sizes, permissions, and creation/modification times

Resources

- Android Cookbook, Ian Darwin, O'Reilly Press, 2017.
- Programming iOS 14 - Dive Deep into Views, View Controllers, and Frameworks, Matt Neuburg, O'Reilly Media Inc., 2021.
- https://developer.android.com/guide/components/processes-and-threads
- https://developer.android.com/topic/security/best-practices
- https://developer.android.com/training/articles/security-tips
- https://source.android.com/security/features
- https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/Multithreading/Introduction/Introduction.html
- https://developer.apple.com/library/content/documentation/Security/Conceptual/Security_Overview/Introduction/Introduction.html
- https://manuals.info.apple.com/MANUALS/1000/MA1902/en_US/apple-platform-security-guide.pdf