Android Internals and Embedded

Course 203 – 40 Hours

Overview
Android provides a rich application framework that allows you to build innovative apps and games for mobile devices in a Java and native languages. We can also find Android OS on many embedded systems as a good replacement to Embedded Linux. Using android as an “improved Linux” saves time because you get all you need out of the box (libraries, packages, data services, etc.)
This course covers the internal side of android from top to bottom. You’ll learn how to use android in an embedded environment, how to access hardware and handling interrupts all over the android stack, and how data flows from the java application down to the kernel driver and the hardware.

Course Objectives

- Understanding the android stack
- Learn how to use android effectively in embedded/real-time systems
- Learn how to add kernel module in android
- Learn how to add native components
- Learn how to add system services
- Learn how to create a system/user app

Who Should Attend
The course is designed for android developers (also beginners) who want to learn android internals and to create and customize an android ROM, and embedded Linux developers who want to migrate to an embedded system on android.

Prerequisites
Delegates should have a working knowledge in C/C++/Java/C#. (at least one)

Course Contents

Android Overview
- Overview
- Android history
- Android stack
- Writing Applications
- components
  - Activities
  - Intents
  - Broadcast receivers
  - Content Providers
- GUI basics, resources

- manifest file
- Processes and threads
- Examples

Android Stack

- Android Linux Kernel Layer
  - Binder
  - Ashmem
  - Pmem/ION
  - Wakelock
  - Early Suspend
  - Alarm
  - Low Memory Killer
  - Logger
  - Alarm
  - Paranoid Network Security
  - Other Kernel Changes
  - Real time requirements and RT-patch

- Android User-Space Native Layer
  - Overview
  - Bionic (libc)
  - User-space Hardware Abstraction Layer (HAL)
  - Native Daemons: ueventd, servicemanager, vold, netd, rild, mediaserver, keystore, racoon, zygote, system_server, adbd, surfaceflinger, etc.
  - Function libraries: libwebcore (Web Kit), V8, SQLite, libssl (OpenSSL), etc.
  - Android Runtime / Dalvik Virtual Machine
  - Real – time requirements

- Android Application Framework Layer
  - Overview
  - Managers and Services
  - Overview of system services and how to use them
  - The binder IPC
  - The system server and service manager

Android Native Development Kit (NDK)

- What is in NDK?
- Why NDK?
- Java Native Interface (JNI)
- Using NDK
- NDK and JNI by Example
- NDK’s Stable APIs

Inter Process Communication (IPC) with Android Binder and AIDL

- Why IPC?
- What is Binder?
- What is AIDL?
- Building a Binder-based Service and Client by Example
- Async-IPC via Binder by Example

Android Security

- Android Security Architecture
- Application Signing
- User IDs
- File Access
- Using Permissions
- Permission Enforcement
- Declaring Custom Permissions
- Custom Permissions by Example
- Encryption in java and native code

Android Startup

- Bootloading the Kernel
- Android's init Startup
- Startup of daemons
- Zygote Startup
- System Server Startup
- Startup of system services
- Startup of applications

Customizing Android

- Setting up Custom Device
- The Build System
- Adding a Custom Kernel
- Adding a custom driver
- Integrating with ueventd
- Adding a Custom Native Library and Executable
- Exposing our Native Library via Java (i.e. JNI)
- Consuming our a Custom Java/JNI, Native Library via a Custom App
- Exposing our Custom Library via a Custom IPC/Binder Service
- Building a Custom App Using a Custom Service Manager
- Creating a full system service
- Handling interrupts at any layer
- Debugging code in AOSP (application and libraries)