Google Android Platform
Introduction to the Android API, HAL and SDK

Zhaohui Wang, Angelos Stavrou
zwange@gmu.edu, astavrou@gmu.edu
George Mason University
What is Android?

“Android delivers a complete set of software for mobile devices: an operating system, middleware and key mobile applications.”

-- http://android.com/about/
What is Android?

- A software stack, and nothing more
- Android was first released on Nov 12, 2007
- Latest Release, Android 2.1 on Jan 12, 2009, with Google’s Nexus One smart phone device
- MOST of the code under Apache License
- Linux based kernel, now on 2.6.29
- ARM based MSM (Qualcomm) chipset
- Graphics, Audio and other HAL implementations
What is Android?

- Development, debugging tools
- Dalvik JVM, (http://www.dalvikvm.com/)
- SDK available on 3 major OSes
- Incomplete/non standard GNU libraries/utils
- Includes “key mobile applications”, Google’s service highly integrated
The Genesis of Android?

- Open Handset Alliance:
  
  http://www.openhandsetalliance.com/

  - Google, eBay, OMRON, PacketVideo, ...
  - ASUSTeK, HTC, LG, Garmin, Motorola, ...
  - Sprint Nextel, T-Mobile, ...
  - ARM, Atheros, Broadcom, Qualcomm, TI, ...

To date, more than 47 organizations
Noteworthy Features

- Android uses Java:
  - ... Everywhere, but only the mobile-appropriate bits!
  - “Android is almost but not quite Java(tm)”

- And so will you:
  - But nothing prevents native processes
  - Some native interfaces are available

- Broad Java support:
  - java.io;
  - java.security;
  - java.net;
  - java.sql;
Noteworthy Features

- **Strong security:**
  - Permissions-based
  - Applications sandboxed in separate VMs
  - Pervasive use of Linux process model

- **Built-in SQL:**
  - Property storage, retrieval
  - Utilized by nearly all standard components
  - Preferred, but not required

- **Specialized APIs:**
  - SurfaceFlinger
  - AudioFlinger
Noteworthy Features

- Highly-optimized Java implementation:
  - “Dalvik” VM implemented by Google
  - Custom bytecode format, processor model
  - Register-based, not stack-based

- Why?
  - “Didn’t want to pay Sun” (probably untrue)
  - Very memory- and performance-efficient
  - Highly tuned to limitations of small hardware

- Centralized object lifetime management:
  - Tied to component model
  - Tied to process model
  - Tied to user interface model
  - Tied to security model
Basic Terminology

- **Activity**: 
  - A single visual user interface component 
  - List of menu selections, icons, checkboxes, ... 
  - A reusable component 

- **Service**: 
  - “Headless” activity component 
  - Background processes 

- **Application**: 
  - Sequence of one or more Activities 
  - Manifest tells which Activity to run first 
  - Activities might come from other applications 
  - Not the Linux concept of “application”!
Basic Terminology

- **Task stack:**
  - Sequences of application-centric Activity classes
  - Foreground is visible to user
  - BACK key returns to most-recent Activity

- **Broadcast receiver:**
  - Component that receives announcements
  - No user interface
  - May launch an Activity in response

- **Content provider:**
  - Provides application data to others
  - The only way to share data
Power Management

- Obviously important!
  - Can be a difficult problem to solve
  - Too much model exposure is bad
  - Too little is also bad

- Extends the Linux device model:
  - Introduces “wake locks”
  - See android.os.PowerManager

- In a nutshell:
  - Applications don’t control power at all
  - Applications hold “locks” on power states
  - If no locks are held, Android powers down
Power Management

- **PARTIAL_WAKE_LOCK**
  - CPU on, screen off, keyboard off
  - Cannot power down via power button

- **SCREEN_DIM_WAKE_LOCK**
  - CPU on, screen dim, keyboard off

- **SCREEN_BRIGHT_WAKE_LOCK**
  - CPU on, screen bright, keyboard off

- **FULL_WAKE_LOCK**
  - CPU on, screen on, keyboard bright
Power Management

- **Example**

1. PowerManager pm =
2. (PowerManager) getSystemService(Context.POWER_SERVICE);
3. PowerManager.WakeLock wl =
4. pm.newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK, "tag");
5. wl.acquire();
6. // ..screen will stay on during this section..
7. wl.release();
Audio and Video APIs

- MediaPlayer class:
  - Standard support for many data formats
  - URI invokes appropriate input method
  - Consistent API regardless of data source

- MediaRecorder class:
  - Support for audio recording only
  - Video recording is “planned”

- Surfaceflinger:
  - Centralized framebuffer management
  - Related to 2D h/w acceleration

- Audioflinger:
  - Centralized audio stream management

You don’t work with these flingers directly!
Audio and Video APIs

Example

1. MediaPlayer mp = new MediaPlayer();
2. mp.setDataSource(PATH_TO_FILE);
3. mp.prepare();
4. mp.start();
5. mp.pause();
6. mp.stop();
Android Architecture

- **Applications**
  - Home
  - Contacts
  - Phone
  - Browser
  - ...

- **Application Framework**
  - Activity Manager
  - Window Manager
  - Content Providers
  - View System
  - Package Manager
  - Telephony Manager
  - Resource Manager
  - Location Manager
  - Notification Manager

- **Libraries**
  - Surface Manager
  - Media Framework
  - SQLite
  - OpenGL | ES
  - FreeType
  - WebKit
  - SGL
  - SSL
  - libc

- **Android Runtime**
  - Core Libraries
  - Dalvik Virtual Machine

- **Linux Kernel**
  - Display Driver
  - Camera Driver
  - Flash Memory Driver
  - Binder (IPC) Driver
  - Keypad Driver
  - WiFi Driver
  - Audio Drivers
  - Power Management
Android Package System

- APK files:
  - Package manifests
  - Classes
  - Dalvik bytecodes
  - Signatures, if any
The Hardware

- **CPU**: Qualcomm QSX8250 1Ghz
- **Mother board**: Qualcomm Mobile Station Modem (MSM) chipset, MSM7k series
- **RAM**: 512 MB
- **ROM**: 512 MB, partitioned as boot/system/userdata/cache
- **External Storage**: 4GB micro SD
- **Audio Processor**: msm_qdsp6 onboard processor, Firmware at /system/etc/vpimg
The Hardware

- Camera: Sensor_s5k3e2fx, 5 MegaPixels
- Wifi+BlueTooth+FM: Boardcom BCM 4329, 802.11a/b/g/n, firmware at /system/etc/firmware/fw_bcm4329.bin
- Touch Screen Input: msm_ts touchscreen controller
- Vibrator: Msm_vibrator on board vibrator
- Digital Compass: AK8973
The Hardware
System Initialization

- Bootloader: HBOOT-0.33.0012
- RADIO-4.02.02.14

1. kernel
2. Init.mahimahi.rc  init.rc
3. debuggerd
4. AndroidRuntime
5. CameraService
6. System server(NetStat, Connectivity, WifiService, etc)
7. Zygote
8. Apps
General procedure:

- Get the code
  - 2.1GB (!) of git trees
  - Uses the repo tool to manage
- Build it
- Install it
- Tweaking and add your own code
- Build it and test it

- [http://android.git.kernel.org/](http://android.git.kernel.org/)
Building the Android Runtime

- `# repo init -b eclair[donut|cupcake] -u git://android.git.kernel.org/platform/manifest.git`
- `# repo sync`
  - ...wait for 2.1GB code downloading
  - ... apply tweaks ...
- `# make [TARGET_PRODUCT=generic]`
Installing Android into a Target

- Build products:
  - userdata.img
  - ramdisk.img
  - system.img
  - kernel.img/boot.img

- And also:
  - out/target/product/<name>/root
  - out/target/product/<name>/system
  - out/target/product/<name>/data
Installing Android into a Target

- “What’s in there?”
  - The Android filesystem

- # ls root

```
data/    init  init.rc  sys/
default.prop  init.goldfish.rc  proc/  system/
dev/  initlogo.rle  sbin/
```

- # ls system

```
app/  build.prop  fonts/  lib/  usr/
bin/  etc/  framework/  media/  xbin/
```
The Android SDK

- Key components:
  - Compilers, other tools
  - Documentation
  - Examples
  - Hardware emulator
  - Android Debug Bridge (adb)

Debugging your first Android App

- Configure USB connection, if you are working with devices

- Test adb and connect to device
  N:\android-sdk-windows\tools>adb devices
  * daemon not running. starting it now *
  * daemon started successfully *
  List of devices attached
  HT9CNP804091 device
  emulator-5556 device

- Launch a shell via adb:
  - The shell is actually on the target!
  N:\android-sdk-windows\tools>adb shell
  $

Debugging your first Android App
Eclipse Android Plugin

Android Development Tool (ADT):
- Custom plugin for Eclipse IDE

Helps automate:
- Set up new Android projects
- Create new applications, components
- Debugging
Eclipse Android Plugin

- Install Eclipse, then:
  - Click Help | Software Updates...
  - https://dl-ssl.google.com/android/eclipse/
  - Click Install...

- Then:
  - Point Eclipse to the Android SDK directory
  - Window | Preferences | Android

Your task

I. Get your helloworld running

II. Profiling and tracing your app
Project Ideas

- Adore-ng rootkit porting on Android architecture
- Iphone OS Security Anatomy
- Iphone/Android MP3 decoder local exploit
- Your Idea (Brainstorming)
Recommended Readings

- Understanding Android's Security Framework
  http://siis.cse.psu.edu/android_sec_tutorial.html

  A very good tutorial at CCS2008:

- Mobile application Security on Android

- Android Developer Lab Tour,
  Washington D.C, Feb, 9, 2010