Android
Background

• Completely open source
  • What does this mean for consumption?
• Mobile Focused
  • What constraints does this add?
• Based on Linux kernel
How it stacks up

Applications

Runtime

Linux Kernel
How it stacks up

Custom stripped down version of Linux 2.6 (2003)
Re-use for existing drivers (wifi, display, ... etc)
NO GAURENTEE THINGS THAT WORK ON LINUX WORK ON ANDROID
How it stacks up

- Applications
- Runtime
  - ART/Dalvik Virtual Machines
  - libc (Bionc and not posix)
- Linux Kernel
  - Custom stripped down version of Linux 2.6 (2003)
  - Re-use for existing drivers (wifi, display, ... etc)
  - NO GAURENTEE THINGS THAT WORK ON LINUX WORK ON ANDROID
How it stacks up

Linux Kernel

Custom stripped down version of Linux 2.6 (2003)
Re-use for existing drivers (wifi, display, ... etc)
NO GAURENTTEE THINGS THAT WORK ON LINUX WORK ON ANDROID

Runtime

ART/Dalvik Virtual Machines
libc (Bionc and not posix)

Applications

Apps written in C or Java (Google java, not oracle’s)
How it stacks up
(Google’s complete version)
Memory Model

• Constraint: Low memory, low battery, low/slow storage
• Issue: What do we do when we run out of physical memory?
• What happens with our current knowledge?
Memory Model

• Constraint: Low memory, low battery, low/slow storage
• Issue: What do we do when we run out of physical memory?
• What happens with our current knowledge?
  • Swap pages: Use disk as a secondary storage to write a page of memory out to
  • Issue for Mobile?
Memory Model

• Constraint: Low memory, low battery, low/slow storage
• Issue: What do we do when we run out of physical memory?
• What happens with our current knowledge?
  • Swap pages: Use disk as a secondary storage to write a page of memory out to
  • Issue for Mobile?
• Android App Life-cycle model
Memory Model

• Take advantage of only one app open on screen
• When users need more resources, kill an old app
• Give programmers a signal to tell them they are going to die to manually save any state
• Restart Process when brought back to foreground
Security Model

• Motivation: Application isolation – never allow a different app to read my app’s data
• Solution: So many things...
Security Model

• Motivation: Application isolation – never allow a different app to read my app’s data

• Solution: So many things...

• App ownership: Each process is it’s own user, all app’s children of “Zygote” process

• Hardware/Memory Protection: All access coordinated through OS and runtime. Everything is sandboxed so the OS chooses what it is allowed to share

• Permissions: User must consent
From code to processor
Standard Linux Executable

- C code
- Compile to machine specific instructions
- Fork new process
- Load instructions into new process
- Add process to scheduler
- Running
Android – Off device

- Applications Written In Java using standard SDK, or C/C++ using NDK
- Compiled using gradle (or other) tools. Creates .apk file
  - This is basically a .class file for java
  - Runs compiler optimizations and condenses a bunch of .dex files together
  - .dex is an architecture independent file
  - Example:
    - dump
    - getNativeStuff() is example of NDK method
Android – On device

• Two different runtimes to interpret .apk files ART & Dalvik

• Dalvik is the legacy system
  • Slow!! Interprets virtually all instructions just like java would (Just In Time JIT compiler)
  • Froyo tried to solve this issue by constantly profiling and saving the compilation of common code paths

• Android Runtime introduced first in KitKat, Exclusive runtime in Lolipop
  •Ahead of time compilation: Slower to install apps, requires more storage
Android – On Device

Dalvik

- Decompose .apk into .dex files
- “Zygote” starts App

ART

- Decompose .apk into .dex files
- Compile .dex files to .oat files
  - Sample dump
- “Zygote” starts App
Sidebar: Zygote

• One of the first Processes started on the device
  • Definition of Zygote: “It is the initial cell formed when a new organism is produced”

• Creates app processes

• Handles all the heavy lifting of mapping memory
  • Sets up what is loaded where, and what pieces of physical memory can be shared between all apps... etc
Android – On Device

Dalvik
• Decompose .apk into .dex files
• “Zygote” starts App
• In the scheduler, .dex instruction loaded
• Instruction translated to machine code
• Run code

ART
• Decompose .apk into .dex files
• Compile .dex files to .oat files
  • Sample dump
• “Zygote” starts App
• In the scheduler, .oat instruction loaded
• Run code
Android – On device

• Check out /proc/[pid]/ folder! A lot of cool things