Android Through a Linux Lens

Jonathan Levin

http://Technologeeks.com - @Technologeeks

http://NewAndroidBook.com

Ready your Droids

- It helps to follow along and try some hands-on
  - (this is, after all, a tutorial, and not just another lecture)
- If you have a real device – great
  - But advanced tracing/debugging does need root access
- At a minimum, fire up an L emulator, and adb to it.

What this isn't

- An ADB shell primer
- A CLI how-to
- A native-level/NDK how-to
- A debugging primer for Dalvik/DDMS

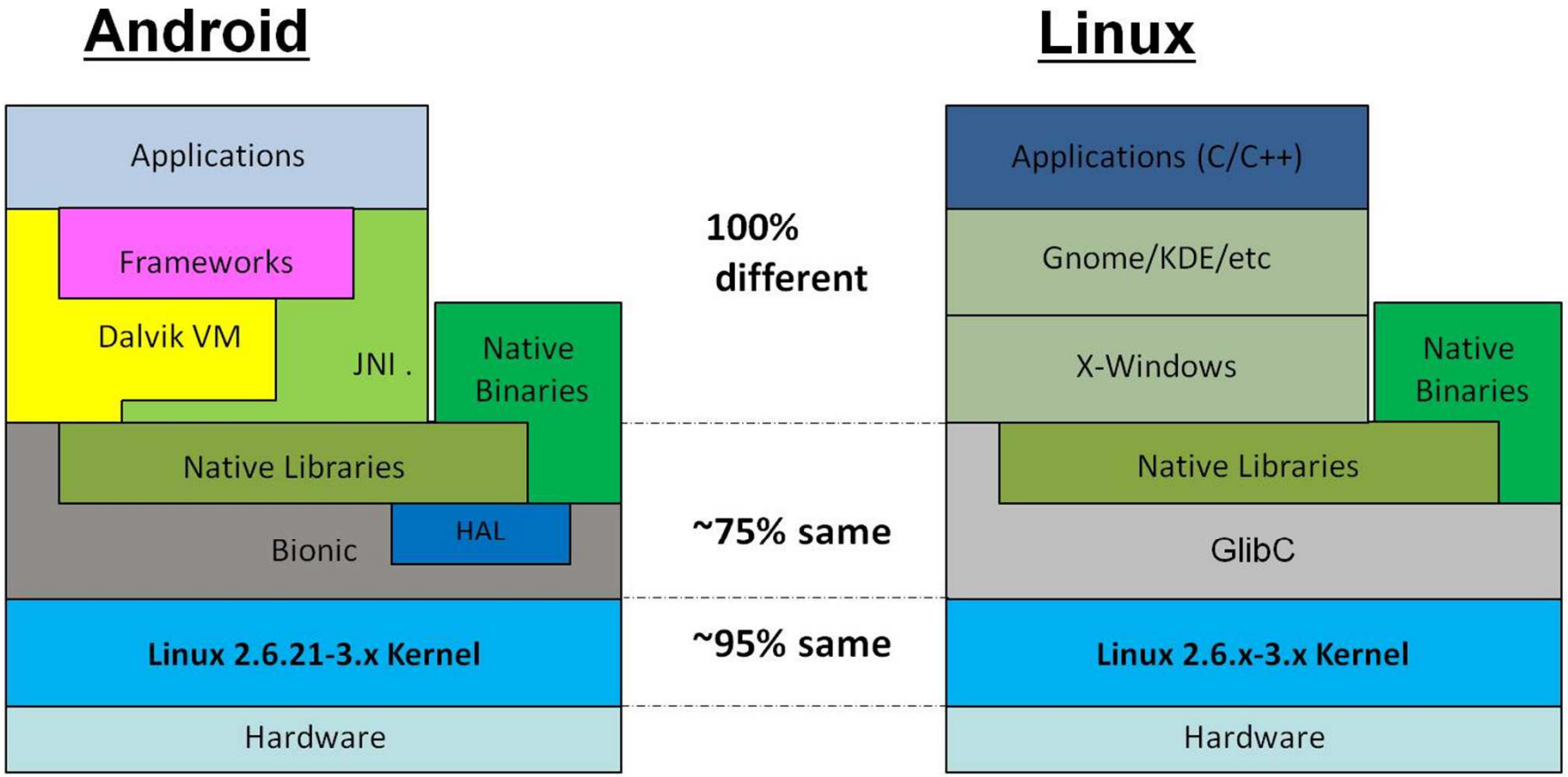
What this is

- Collection of native and CLI level debugging techniques
- Uses AOSP-supplied tools, and Linux facilities
- Actually also usable for Linux native level debugging
- An excerpt from my book (Volume I, Chapter 7)

The Book

- "Android Internals: A Confectioner's Cookbook"
  - **深入解析Android 操作系统 - Coming in Chinese (by Dec 2015)**
- Parallels "Mac OS X and iOS Internals" (but for Android)
  - BTW MOXiI is getting a 2nd Edition (10.10/iOS 8) – Nov 2015!
- Volume I is available! Updated for M PR1
  - M changes require rewrite for Volume II
- http://newandroidbook.com/
  - FAQ, TOC and plenty of bonus materials
  - Check out technologeeks.com courses!

Android vs. Linux



Everything is a process

- Apps, system daemons, utilities are all ***processes***.
- Android provides Lifecycle illusions
- Underlying app utilitizes several threads
  - Main thread is UI Thread, runs App Looper
  - Ancillary threads support runtime services
  - Additional threads may be created by developer

- Linux perspective won't see Dalvik/ART runtime
  - But will see what the threads are doing!

ADB and the shell

- ADB provides a command line shell as uid shell

```
shell@htc_m8wl:/ $ id
uid=2000(shell) gid=2000(shell) groups=1003(graphics),1004(input),1007(log),
1009(mount),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin),
3002(net_bt),3003(inet),3006(net_bw_stats) context=u:r:shell:s0
```

- Shell is MirBSD Korn shell, with scripting abilities

- Recommendation: Install SSHD (or dropbear, etc)
  - Frees you from tethering requirement, fully remote
  - Allows easier (and safer) root access
  - Will require public key authentication only (no password..)

ToolBox vs. BusyBox

- Most CLI commands implemented via toolbox
- Toolbox is Android specific subset of busybox
  - Pros: linked with Bionic, recognizes AIDs
  - Cons: limited toolset, partial functionality
- Recommendation: Install BusyBox
  - Statically linked binary, so no dependencies
  - Can compile from source, but plenty of binaries out there
- M provides “toybox” with more tools (try –a)

Getting tools to your device

- A lot of the tools you need are right in the emulator
  - Most in /system/xbin, and not present in most devices
- Can adb pull from emulator image, push to device
- Android version should match
- Remember to move libraries as well!
  - Find dependencies using objdump –x | grep NEEDED

Using what's already there

- Linux provides three pseudo file systems:
  - /proc
  - /sys
  - /sys/kernel/debug

pseudo- | ˈsudoʊ | (also **pseud-** before a vowel)
comb. form
1 supposed or purporting to be but not really so; false; not genuine: *pseudonym* | *pseudoscience*.
2 resembling or imitating: *pseudohallucination* | *pseudo-French*.

ORIGIN from Greek ***pseudēs 'false,' pseudos 'falsehood.'***

- NOT real files – just on demand when you poll them

The procfs (/proc) filesystem

- A plethora of diagnostic information:
  - General system diagnostics (/proc root)
  - Subsystem information (/proc/bus, /proc/irq, …)
  - Sysctl variables (/proc/sys)
  - Per process diagnostics (/proc/[0-9]*])

The sysfs (/sys) filesystem

- Complements /proc, and provides:
  - Hardware and device representations
  - Kernel module information and parameters (/sys/module)
  - Kernel subsystem control
- Useful point of departure: vendor HAL modules
  - Contain hard coded paths of devices
  - Can also consult init.rc in most cases (pref. Init.*platform*.rc)

The debugfs (/d) filesystem

- Exclusively for kernel-level debugging and control
  - Kernel ftrace functionality
  - Binder debugging
  - Android atrace

```
root@Android:/ # mount -t debugfs none /sys/kernel/debug
root@Android:/ # mount | grep debug
none /sys/kernel/debug debugfs rw,relatime 0 0
```

- Usually mounted as `/sys/kernel/debug`, symlinked `/d`

ftrace

- Hands down, most useful tracing functionality EVER
- Actual tracers depend on what was compiled in kernel:

Tracer	Performs
nop	Absolutely nothing (default)
Block	Block device activity
function	In kernel function calls
function-graph	Full flow tracing (awesome!)

- Spoiler: your default kernel may only have “nop”...
  - .. But you can recompile the kernel easily! (And it’s worth it!)

ftrace

- Don’t despair! Even with “nop” you still have... Events!

Events	Traces
binder	Binder IPC calls (useful!)
ext4/f2fs	Low level filesystem activity
block/mmc	Lower level block/MMC I/O
Power	Power management events
Cpufreq	Governors
Sys_calls/raw_syscalls	System calls

keychords

- Little known feature of /init
- Binds services/commands to key combination
  - “keys” are physical buttons on device, as Android codes
- Uses `/dev/keychord`, where available
- Specify “keycodes” combination in /init.rc or other rc
- Even more useful when a physical keyboard is present

Activity Diagnostics

Activity Diagnostics

- Tracing = monitoring run time activity of process

- Uses:
  - performance benchmarking
  - Logging and monitoring resource access

Activity Diagnostics - /proc

- A cornucopia of per process related information:

/proc entry	Provides
/proc/$pid/cwd	Symbolic link to current working directory
/proc/$pid/cmdline	NULL separated argv[] of process
/proc/$pid/fd	Directory with symbolic links to open descriptors
/proc/$pid/fdinfo	Information about open descriptors
/proc/$pid/status	Human readable general statistics (VM + More)
/proc/$pid/task	Directory per thread
/proc/$pid/wchan	Wait channel (indicates kernel syscall block/sleep)

Activity Diagnostics - /proc

- Iterating over task/* will show threads
- Threads largely have same stats, save:
  - Status/Name – Dalvik threads are named with prctl(2)
  - wchan – Kernel wait channel/syscall
- You can grep name $t/status to isolate threads
- You can then trace or suspend specific threads

Activity Diagnostics - /proc

Output 7-5: The annotated /proc/*pid*/status entry

```
shell@flounder:/proc $ cat /proc/511/status
Name:   system_server  # Same as /proc/511/comm
State:  S (sleeping)   # or R - running, T - Stopped, D - Uninterruptible (deep) sleep
Tgid:   511            # Thread Group id: The real process id
Pid:    511            # Thread, not process id
PPid:   211            # Parent Thread Group id
TracerPid:      0      # Any ptrace(2) attached process, like strace, gdb or debuggerd
Uid:    1000    1000    1000    1000  # Real, Effective, Set and File-System UIDs
Gid:    1000    1000    1000    1000  # Real, Effective, Set and File-System GIDs
FDSize: 2048                          # Maximum # of file descriptors allowed
Groups: 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1018 1032 3001 3002 3003 3006 3007
VmPeak:  2367448 kB      # Virtual Memory high-water mark
VmSize:  2258636 kB      # Virtual Memory size, present
VmLck:         0 kB      # Memory locked by mlock(2) APIs
VmPin:         0 kB      # Pinned memory
VmHWM:    178100 kB      # RSSPeak - i.e. Resident memory footprint high-water mark
VmRSS:    151600 kB      # Resident memory footprint, present
VmData:   230448 kB      # Size of data segment (heap memory)
VmStk:      8192 kB      # Size of process thread stacks
VmExe:        16 kB      # Size of executable
VmLib:    120016 kB      # Memory used by shared library (.so) files
VmPTE:       968 kB      # Memory used by Page Table Entries
VmSwap:        0 kB      # Memory used by process in swap (If no swap, always 0)
Threads:      82         # Number of threads. If > 1, this is a multi-threaded process
SigQ:   3/6826           # Handled Signals/Size of signal queue
SigPnd: 0000000000000000 # Bitmask of pending signals (for thread)
ShdPnd: 0000000000000000 # Shared pending signals for process
SigBlk: 0000000000001204 # Bitmask of signals blocked (by SIG_BLOCK)
SigIgn: 0000000000000000 # Bitmask of signals ignored (by SIG_IGN)
SigCgt: 00000002000094f8 # Bitmask of signals caught (by handlers)
CapInh: 0000000000000000 # Bitmask of inherited capabilities
CapPrm: 0000001007813c20 # Bitmask of permitted capabilities
CapEff: 0000001007813c20 # Bitmask of effective capabilities
CapBnd: 0000000000000000 # Bitmask of capabilities bounding set
Cpus_allowed:   3        # Bitmask of CPUs allowed (3 = 0011)
Cpus_allowed_list: 0-1   # List of CPUs allowed, for the hex-challenged
voluntary_ctxt_switches:        40684 # Voluntary (system call induced) context switches
nonvoluntary_ctxt_switches:     13471 # Nonvoluntary (preemption induced) context switches
```

Activity Diagnostics - Tools

- AOSP provides the lsof tool to list open files
  - Not just files, but actually any file descriptor for process
- Extremely useful with grep to isolate files

Activity Diagnostics - Tools

- AOSP also provides the strace binary to trace syscalls
  - Hands down, the #1 debugging tool out there
  - Based on `ptrace(2)` API, no dependencies

- Useful in oh-so-may ways:
  - Can actually parse and present system call arguments
  - Can follow forks and threads
  - Can be used for timing of syscalls
  - Can introduce artificial latency(!)

Activity Diagnostics - Tools

- The `ltrace` tool can also be ported to Android
  - Similar to `strace`, but provides **library call** information
  - Uses `ptrace(2)`, but a lot heavier, and needs libelf.

- Supplements `strace` when your problem is in a lib:
  - Arguments and features similar to `ltrace`
  - Can also be used for syscalls (with –S)

- http://NewOSXBook.com also offers `jtrace`

Activity Diagnostics - Triggers

- Linux doesn't really support file access triggers
  - Inotify is an exception, but no shell command for it*
  - You have to be specific on which file/directory to watch
  - Still no notification for in-file access (say, certain offset)
- Using /proc and a little bit of scripting, however…

Listing 7-1: A small script to watch and act on a file position in a process

```
PID=$1                      # PID is first argument
FD=$2                       # FD to watch is second argument
OFFSET=$3                   # OFFSET to monitor is third argument
CUT_COMMAND='busybox cut'   # needed because toolbox doesn't have cut built-in

# This isolates just the numerical offset from the fdinfo entry of $FD

#      Get the data               | isolate pos line  | isolate numerical value
CUROFF=`cat /proc/$PID/fdinfo/$FD |     grep pos      | $CUT_COMMAND -d':' -f2`

if [[ $CUROFF -gt $OFF ]]; then
  echo Do something
 # Insert command to execute on trigger here
else
  echo Nothing to do.
fi
```

* - But check out http://NewAndroidBook.com/files/inotify.tar, also M toybox inotify

Memory Tracing and Debugging

Memory Diagnostics

- RAM is the most important resource in Android
- Applications leave in perpetual fear of OOM/LMK
- Most memory in Android is shared when possible
- Important to understand memory diagnostics

Low Memory Killer

- Protector of all droids, sworn adversary of all apps
  - Linux OOM is not-so-deterministic.
  - LMK more predictable – but more conservative
- Each process has an oom_score and an oom_adj
  - Native apps can cheat death – Dalvik ones can't (LMKd)
- LMK parameters can be tweaked through sysfs

/sysfs entry	Provides
…adj	Array of oom_adjs to kill on minfree hit
…minfree	Array of 4k multiples to start kill adj on
cost	Memory shrinker cost
debug_level	Verbosity for kill operations (1-5)

Memory Diagnostics

- VSS: Virtual Set Size (a.k.a VMSize)

- RSS: Resident Set Size

  USS = Unique Set Size + ShSS = Shared Set Size

  HWM = RSS Peak

- PSS: Proportional Set Size

  USS + (ShSS/#Shares)

Memory Diagnostics - /proc

- /proc filesystem provides key memory statistics:
  - Systemwide:

/proc entry	Provides
/proc/meminfo	Global memory statistics
/proc/vmstat	Kernel view of global memory, per variables

  - Per process:

/proc entry	Provides
/proc/$pid/maps	Address Space Layout (mmapped and anon)
/proc/$pid/smaps	As maps + per mapping information
/proc/$pid/status	VM Statistics, and much more

Memory Diagnostics - tools

- AOSP provides `procrank` and `librank` tools:
  - `procrank`: Ranks processes by memory utilization
  - `librank`: Ranks libraries by memory utilization (sharing)

- KitKat and later provide the `memtrack` tool
  - Logs memory utilization to Android logs

Framework Level Debugging

UpCall Scripts

- Android provides several CLI interfaces to Dalvik

script	Usage
am	Interact with ActivityManager. Start activities, fire intents and much more.
appwidget	L: Grant/Revoke User Application Widgets
bmgr	Backup Manager Interface
bu	Start backup
content	Interface to Android's content providers
dpm	M : Device Admin/Profile Management
ime	Control Input-Method-Editors
input	Interact with InputManager, inject input events (discussed in Volume II).
media	Control the current media client (play/pause/etc)
monkey	Run an APK with randomly generated input events
pm	Interact with PackageManager, list/install/remove packages, list permissions, etc.
requestsync	Sync accounts
settings	Get/set system settings
sm	M: Storage Management
svc	Control the power , data, wifi and USB services
uiautomator	Performs UI Automation tests, dumps view hierarchy, etc.
wm	Interact with WindowManager, change display size/density, etc.

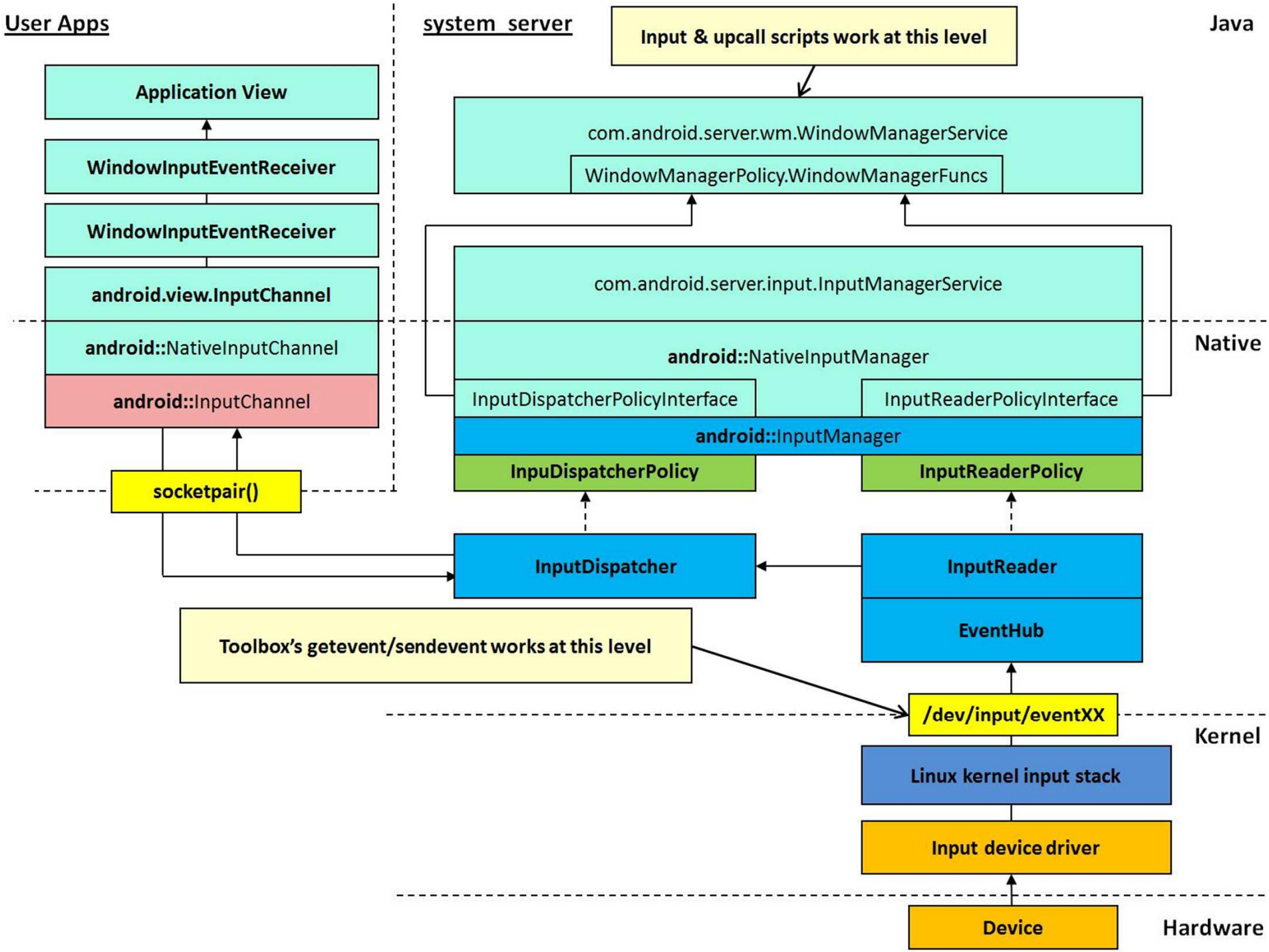
The service tool

- CLI interface to `servicemanger` and the Binder
- Simple, but powerful
- Can automate virtually all Android services
  - Does require root access for some of the services
- Depends on service called, virtually undocumented*
  - Can get service calls through AIDL mappings in AOSP

* - Well, until ***now***.. That's why I wrote the book - http://NewAndroidBook.com/

Input Events

- Toolbox's getevent/sendevent can automate input
- The input upcall script can inject to input manager
- UIAutomator handles more complex events
- Monkey just goes wild all over the place
- For more details, q.v. Andevcon session on Fri 14:45


User Apps
system_server
Java
Input & upcall scripts work at this level
Application View
com.android.server.wm.WindowManagerService
WindowManagerPolicy.WindowManagerFuncs
WindowInputEventReceiver
WindowInputEventReceiver
com.android.server.input.InputManagerService
android.view.InputChannel
android::NativeInputChannel
android::NativeInputManager
Native
android::InputChannel
InputDispatcherPolicyInterface
InputReaderPolicyInterface
android::InputManager
InpuDispatcherPolicy
InputReaderPolicy
socketpair()
InputDispatcher
InputReader
EventHub
Toolbox's getevent/sendevent works at this level
/dev/input/eventXX
Kernel
Linux kernel input stack
Input device driver
Device
Hardware

Post Mortem

Post Mortem Debugging

- Android doesn't support core dumps by default
  - Storage space is limited, and cores can be pretty big
  - `ulimit -c 0` is set in /init (via setrlimit) and inherited

- Tombstones used instead of cores
  - Application crashes, debuggerd is notified
  - Checks if `debug.db.uid` property is set, to wait for `gdb`
  - Otherwise, engraves "tombstone" (crash report)

Tombstones

- Debuggerd uses Linux's ptrace(2) API to:
  - Enumerate all threads
  - Get register state for each thread
  - Get Stack trace for all threads
  - Get stack and instruction pointer memory contents
- Tombstone data is highly architecture specific

If you *do* want cores..

- Set ulimit to unlimited for crashing process
  - If you actively set to 0, must be root to unset
- Modify /proc/sys/kernel/core_pattern
  - Specify filename, can use %h, %e, %u, %p
  - Can also specify pipe (|) and command name
    - Command accepts core via stdin (e.g. HTC's dalvik_coredump.sh)

You've been watching...

Android Through a Linux Lens

Jonathan Levin

http://Technologeeks.com - @Technologeeks

http://NewAndroidBook.com