The Android Input Architecture

The journey of a thousand function calls starts with an Interrupt

Jonathan Levin
Technologeeks.com (@Technologeeks)
NewAndroidBook.com
About This Talk

• Discusses the Android Input Stack, in depth

• Follows flow of input (up to App, sans IME)
  – Avoids code as much as possible

• Demonstrates a few handy input tools

• Essentially an excerpt from the Book.
The Book

• “Android Internals:: A Confectioner’s Cookbook”

• Unofficial parallel to “Mac OS X and iOS Internals”
  • (which, btw, is coming out in a 2\textsuperscript{nd} Edition for iOS 9/OS X 10.11!)

• Volume I released earlier this year
  – Already updated for Android M PR1-2!

• Volume II to be released soon
  • As soon as Google stabilizes M

• \url{http://www.NewAndroidBook.com/}
  – FAQ, TOC and plenty of bonus materials
What do we know about input?

The activity gets the input as an event, via the target view’s onXXX event callback.

Physical events (e.g. touches, clicks, swipe, etc) occur at the device hardware level.

Activity gets input as part of a specified event callback.
Behind the scenes

- The Android input stack is actually complex
- Input flow involves multiple components
- Requires Inter Process Communication (IPC)
- Plenty of input sources:
  - Touch screen
  - Keyboards (real, virtual)
  - Sensors (accelerometer, GPS, light, temp..)
- Even more on IoT devices (e.g. Treadmills!)
- Not all input consumable by views
The Linux Kernel

- The very first component of the input stack
- Nothing Android specific here
- Delegates input retrieval to device driver
- All Input drivers conform to Linux Input Model

The activity gets the input as an event, via the target view’s onXXX event callback
The Linux Input Model

- interrupt statistics in /proc/interrupts
  (nice bonus: # of active CPUs)
- Drivers claim interrupt (request_irq)
- Driver callback invoked by kernel

CPU responds to interrupts, calls kernel, to dispatch to device driver

All Input starts with some type of interrupt, generated by device
The Linux Input Model

- Driver registers input_device
- Device file created: /dev/input/eventXX
- Driver reports event as an event record

CPU responds to interrupts, calls kernel, to dispatch to device driver

All Input starts with some type of interrupt, generated by device
The Linux Input Model

The adb shell can do all this because it's a member of the input group

```
shell@htc_himaulatt:/ $ ls -l /dev/input
```

```
crw-rw---- root     input     13,  64 2015-07-27 10:14 event0
crw-rw---- root     input     13,  65 2015-07-27 10:14 event1
```

```
shell@htc_himaulatt:/ $ getevent -l
```

```
add device 1: /dev/input/event3
  name: "qpnp_pon"
add device 2: /dev/input/event2
  name: "AK8789_HALL_SENSOR"
add device 3: /dev/input/event0
  name: "h2w headset"
```

```
# Headphone jack (detects insertion)
could not get driver version for /dev/input/mice, Not a typewriter
add device 4: /dev/input/event4
  name: "gpio-keys"
add device 5: /dev/input/event1
  name: "synaptics_dsx"
```

```
# Power button press
/dev/input/event4: EV_KEY   KEY_POWER            DOWN
/dev/input/event4: EV_SYN       SYN_REPORT  00000000
```

```
# Power button release
/dev/input/event4: EV_KEY   KEY_POWER    UP
/dev/input/event4: EV_SYN       SYN_REPORT  00000000
```

```
# Touch
/dev/input/event1: EV_ABS  ABS_MT_TRACKING_ID   00000002
/dev/input/event1: EV_ABS  ABS_MT_POSITION_X    000001d1
```

```
/dev/input/event1: EV_ABS  ABS_MT_POSITION_Y    0000053e
/dev/input/event1: EV_ABS  ABS_MT_TOUCH_MINOR   0000000a
```

```
/dev/input/event1: EV_SYN  SYN_REPORT  00000000
```

```
The getevent tool reads input events
```
The Linux Input Model

- **Kernel**
  - **Input device driver**
    - **Hardware**

Events injected indistinguishable from driver

# Note /dev/event/input devices are also writable!

shell@htc_himaulatt:/ $ ls -l /dev/input

crw-rw---- root   input   13,   64 2015-07-27 10:14 event0

crw-rw---- root   input   13,   65 2015-07-27 10:14 event1
...

# simulate EV_KEY KEYHOMEPAGE DOWN followed by REPORT
shell@htc_himaulatt:/ $ sendevent /dev/input/event5 1 172 1 \
  sendevent /dev/input5 0 0 0

# To simulate home button hold, delay the following line, simulating the UP/REPORT
shell@htc_himaulatt:/ $ sendevent /dev/input/event5 1 172 0;  \
  sendevent /dev/input5 0 0 0

The `sendevent` tool injects input events
The activity gets the input as an event, via the target view’s onXXX event callback

- Apps don’t have permission to input devices
- System_server therefore gets involved
The activity gets the input as an event, via the target view’s onXXX event callback.

- **EventHub**: responsible for raw events
- **InputReader**: reads and “cooks” events
- **InputDispatcher**: Sends to target view
The Event Hub

- Convert raw events (struct input_event) to Android events (per keymap/layout)
  - /system/usr/keylayout
  - /system/usr/keychars

- Also adds/removes devices
  Detects addition/removal via inotify
  Synthesizes DEVICE_ADDED|REMOVED

The activity gets the input as an event, via the target view’s onXXX event callback

(c) 2015 Jonathan Levin - http://NewAndroidBook.com - Distribute Freely, but please do not plagiarize!
The InputReader

- Only client of the Event Hub
- Reads events and “cooks” them
  synthesizes advanced touch events from MT
  uses device input mappers to process events
- Notifies InputListener (Dispatcher) of events
The InputDispatcher

- Gets cooked event from reader
  Reader calls notifyXXX from InputListenerInterface
- Locates target view in registered windows
- Dispatches event to target app

The activity gets the input as an event, via the target view’s onXXX event callback
Dispatching Events

- Views (Windows) create Input Channels
  - IPC performed via UN*X socketpair(2)
- Input Channels registered with Dispatcher
- Dispatcher finds focused Window
- Writes event to its end of socketpair
Dispatching Events

- But why a socketpair?
  - Application expected to send FINISHED
  - Event dequeued only after response
  - No response can lead to dreaded ANR
Policy Upcalls

- Both Reader/Dispatcher consult “policies”
- Policy provided by upcalls to Java layer

Table: iRPC InputReaderPolicy calls

<table>
<thead>
<tr>
<th>Method</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>getReaderConfiguration(outConfig)</td>
<td>Gets the input reader configuration. Used internally when constructing the InputReader</td>
</tr>
<tr>
<td>obtainPointerController(deviceId)</td>
<td>Gets a pointer controller associated with the specified cursor device. The pointer controller is used for mice (to show the pointer) and when the UI is set to debug touches</td>
</tr>
<tr>
<td>notifyInputDevicesChanged(inputDevices)</td>
<td>Notifies Dalvik that devices have changed. This results (eventually) in an android.view.InputDevice object creation.</td>
</tr>
<tr>
<td>getKeyboardLayoutOverlay(inputDeviceDescriptor)</td>
<td>Get Keyboard layout for an inputDeviceDescriptor</td>
</tr>
<tr>
<td>getDeviceAlias(inputDeviceIdentifier)</td>
<td>Get User Alias for device</td>
</tr>
</tbody>
</table>

(c) 2015 Jonathan Levin - http://NewAndroidBook.com/ - Distribute Freely, but please do not plagiarize!
Policy Upcalls

- Both Reader/Dispatcher consult “policies”
- Policy provided by upcalls to Java layer

<table>
<thead>
<tr>
<th>Method</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDispatcherConfiguration()</td>
<td>Gets the input dispatcher configuration from the policy. Used internally when constructing the InputDispatcher</td>
</tr>
<tr>
<td>notifyNR()</td>
<td>Notify Input Monitor an application is not responding</td>
</tr>
<tr>
<td>notifyInputChannelBroken()</td>
<td>Notify Input Monitor channel can no longer be used</td>
</tr>
<tr>
<td>interceptKeyBeforeQueuing()</td>
<td>Intercept key before queuing</td>
</tr>
<tr>
<td>interceptMotionBeforeQueuing()</td>
<td>Intercept motion before queuing</td>
</tr>
<tr>
<td>dispatchUnhandledKey()</td>
<td>Check if process pid with user ID uid may inject input events into other applications. The InputManagerService class handles this one, checking for INJECT_EVENTS permission.</td>
</tr>
<tr>
<td>checkInjectEventsPermission()</td>
<td>Allow policy to filter event. Implemented by an inputFilter in the InputManagerService class.</td>
</tr>
<tr>
<td>filterInputEvent()</td>
<td>Determine if key repeating is enabled</td>
</tr>
</tbody>
</table>

(c) 2015 Jonathan Levin - http://NewAndroidBook.com/ - Distribute Freely, but please do not plagiarize!
- WindowManagerService registers callbacks
- Enables interception/injection of events
The diagram illustrates the architecture of input handling in Android. It shows the flow of input events from the hardware to the application, highlighting the layers and components involved.

1. **User Apps**
   - Application View
   - WindowInputEventReceiver
   - WindowInputEventReceiver
   - android.view.InputChannel
   - android::NativeInputChannel
   - android::NativeInputChannel

2. **Native**
   - socketpair()
   - InputDispatcherPolicy
   - InputReaderPolicy
   - android::InputManager

3. **system_server**
   - com.android.server.wm.WindowManagerService
   - WindowManagerPolicy.WindowManagerFuncs
   - com.android.server.input.InputManagerService
   - android::NativeInputManager
   - InputDispatcherPolicyInterface
   - InputReaderPolicyInterface

4. **Java**
   - InputDispatcher
   - InputReader
   - EventHub

5. **Kernel**
   - Linux kernel input stack
   - /dev/input/eventXX
   - Input device driver
   - Device

6. **Android frameworks**
   - Native Input Frameworks
   - InputFlinger
   - Linux Kernel
   - Vendor Specific

(c) 2015 Jonathan Levin - http://NewAndroidBook.com/ - Distribute Freely, but please do not plagiarize!
Toolbox’s sendevent works at this level

Application View

WindowInputEventReceiver

WindowInputEventReceiver

android.view.InputChannel

android::NativeInputChannel

android::InputChannel

socketpair()

system_server

Input upcall script works at this level

com.android.server.wm.WindowManagerService

WindowManagerPolicy.WindowManagerFuncs

com.android.server.input.InputManagerService

android::NativeInputManager

InputDispatcherPolicy Interface

InputReaderPolicy Interface

android::InputManager

InputDispatcherPolicy

InputReaderPolicy

InputDispatcher

InputReader

EventHub

/dev/input/eventXX

Linux kernel input stack

Input device driver

Device

Hardware

Native

User Apps

Java

(c) 2015 Jonathan Levin - http://NewAndroidBook.com/ - Distribute Freely, but please do not plagiarize!
... We’re not done yet..

setView creates a pipeline
... We’re not done yet..

- Synthesizes new events from unhandled input events
- Processes Key/Pointer events, forwards others
- Dispatches to InputMethodManager
- Basic processing, (almost) always forwards

**Application View**

- WindowInputEventReceiver
- android::NativeInputChannel
- android::InputChannel

- socketpair()

**Synthetic Input Stage**

- ViewPreImeInputStage
- NativePreImeInputStage

- EarlyPostImeInputStage

- ViewPostImeInputStage

- NativePostImeInputStage

**Processes Key/Pointer events, forwards others**

**Process all events, suspends window updating during processing for non-key events**

**Deliver**

- finishInputEvent()
protected int onProcess(QueuedInputEvent q) {
    if (q.mEvent instanceof KeyEvent) {
        return processKeyEvent(q);
    } else {
        // If delivering a new non-key event, make sure the window is
        // now allowed to start updating.
        handleDispatchDoneAnimating();
        final int source = q.mEvent.getSource();
        if ((source & InputDevice.SOURCE_CLASS_POINTER) != 0) {
            return processPointerEvent(q);
        } else if ((source & InputDevice.SOURCE_CLASS_TRACKBALL) != 0) {
            return processTrackballEvent(q);
        } else {
            return processGenericMotionEvent(q);
        }
    }
}
Input debugging

• If you can rebuild AOSP:

<table>
<thead>
<tr>
<th>#define</th>
<th>ALOGD output</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG_INPUT_READER_POLICY</td>
<td>NativeInputManager upcalls from the InputReader</td>
</tr>
<tr>
<td>DEBUG_INPUT_DISPATCHER_POLICY</td>
<td>NativeInputManager upcalls from the InputDispatcher</td>
</tr>
<tr>
<td>DEBUG_FOCUS</td>
<td>Input focus tracking</td>
</tr>
<tr>
<td>DEBUG_INJECTION</td>
<td>Input event injection, via injectInputEvent and</td>
</tr>
<tr>
<td></td>
<td>setInjectionResultLocked</td>
</tr>
<tr>
<td>DEBUG_REGISTRATION</td>
<td>Input channel registration and unregistration</td>
</tr>
<tr>
<td></td>
<td>([un]RegisterInputChannel).</td>
</tr>
<tr>
<td>DEBUG_DISPATCH</td>
<td>Input Dispatcher flow</td>
</tr>
<tr>
<td>DEBUG_HOVER</td>
<td>Hover enter and exit</td>
</tr>
</tbody>
</table>

• Use dumpsys input
• Use jtrace
Moral: Don’t touch your device so much!

• Have respect for your poor CPU has to go through EVERY time!

.. Find more detail in Android Internals::The Developer’s View

• More diagrams/flow tracing
• Only the bare minimum of code excerpts required
• Links/References to latest AOSP sources
• The only alternative to reading the source...