MYTHBUSTERS: ANDROID

Matt Porter
Mentor Graphics
Android platform enablement is a hot topic, everybody seems to want Android on their part/board/system.

We test several Android questions today:
- Is Android “Linux”? What does that mean?
- Does Android “Just Work™”?  
- What/where is the Android community?

Looking at some examples will help us answer these questions.
Porting Android

- Linux kernel
  - Android patches
    - Ashmem
    - binder
    - Android PM
  - Arch support
- Android “distro”
  - AOSP
  - Building
- Deploy!
Bionic

- Bionic is Android’s libc
- Not glibc
- BSD derived
- ARM/x86 support only
- Partial pthreads support
- No SysV IPC support
- No STL support
- Prelink is unique to bionic/Android
No linux-headers package

Makes adding new native binaries to Android an annoyance

Minimal “scrubbed” set of headers

Why?

Results in a lot of this:

diff --git a/libc/kernel/common/linux/uinput.h b/libc/kernel/common/linux/uinput.h
new file mode 100644
index 0000000..827d99d
--- /dev/null
+++ b/libc/kernel/common/linux/uinput.h
Device Node Management

- Sorry, no udev here
- Android’s new init replaces udev…poorly

```c
static struct perms_devperms[] = {
    {"/dev/null", 0666, AID_ROOT, AID_ROOT, 0 },
    {"/dev/zero", 0666, AID_ROOT, AID_ROOT, 0 },
    {"/dev/full", 0666, AID_ROOT, AID_ROOT, 0 },
    {"/dev/ptmx", 0666, AID_ROOT, AID_ROOT, 0 },
    {"/dev/tty", 0666, AID_ROOT, AID_ROOT, 0 },
    {"/dev/random", 0666, AID_ROOT, AID_ROOT, 0 },
    {"/dev/urandom", 0666, AID_ROOT, AID_ROOT, 0 },
    ...
};
```

- Yes, that’s policy hardcoded into the init binary
Hotplug

- No hotplug scripts or udev/hal
- Init/Vold replaces that infrastructure
- Types of hotplug events processed are hardcoded in init

/* this should probably be configurable somehow */
if(!strncmp(uevent->subsystem, "graphics", 8)) {
    base = "/dev/graphics/";
    mkdir(base, 0755);
} else if (!strncmp(uevent->subsystem, "oncrpc", 6)) {
    base = "/dev/oncrpc/";
    mkdir(base, 0755);
} else if (!strncmp(uevent->subsystem, "adsp", 4)) {
    base = "/dev/adsp/";
    mkdir(base, 0755);
} else if(!strncmp(uevent->subsystem, "input", 5)) {
    base = "/dev/input/";
    mkdir(base, 0755);
} else if(!strncmp(uevent->subsystem, "mtd", 3)) {
    base = "/dev/mtd/";
    mkdir(base, 0755);
} else if(!strncmp(uevent->subsystem, "misc", 4) &&
        !strncmp(name, "log_", 4)) {
    ...

Storage devices are not managed by HAL
Replacement is vold
  vold only designed to handle mount/unmount of an MMC subsystem device
  Needs help to handle a USB Mass Storage device

if (!(d = opendir(SYSFS_CLASS_MMC_PATH))) {
    LOG_ERROR("Unable to open '%s' (%m)", SYSFS_CLASS_MMC_PATH);
    return -errno;
}
Android input uses standard Linux Input
EventHub auto-discovers input devices
- At boot
- Upon event queue creation (hotplug usb HID)
Input devices categorized by probing EV_* capabilities
- Keyboard
- Trackball
- Touchscreen
- Mouse (by non-mainline patch from Android-x86)
Key mapping handled using a key layout and key character map infrastructure

Problem: key layout/charmap used is matched by the input device name string

```c
if (err <= 0) {
    // a more descriptive name
    ioctl(mFDs[mFDCount].fd, EVIOCGNAME(sizeof(devname)-1), devname);
    devname[sizeof(devname)-1] = 0;
    device->name = devname;
    strcpy(tmpfn, devname);
    // replace all the spaces with underscores
    for (char *p = strchr(tmpfn, ' '); p && *p; p = strchr(tmpfn, ' '))
        *p = '_';
}
```

// find the .kl file we need for this device
const char* root = getenv("ANDROID_ROOT");
snprintf(keylayoutFilename, sizeof(keylayoutFilename),
    "%s/usr/keylayout/%s.kl", root, tmpfn);

This doesn’t work at all for USB keyboards!
Touchscreen

- Touchscreen support makes no use of tslib
- Touchscreen events from the kernel driver are passed on uncooked directly to the Android “key event queue”

```c
if(ioctl(mFDs[id_to_index(device->id)].fd, EVIOCGABS(axis), &info)) {
    LOGE("Error reading absolute controller %d for device %s fd %d\n",
         axis, device->name.string(), mFDs[id_to_index(device->id)].fd);
    return -1;
}
*outMinValue = info.minimum;
*outMaxValue = info.maximum;
*outFlat = info.flat;
*outFuzz = info.fuzz;
return 0;
```

- This results in kernel drivers being hacked for one-off calibration of absolute events being returned
- Patches exist to add tslib support now
Running Android on Framebuffers with larger resolutions (1024x768+) quickly runs into this:

```java
// create the surface Heap manager, which manages the heaps
// (be it in RAM or VRAM) where surfaces are allocated
// We give 8 MB per client.
SurfaceHeapManager = new SurfaceHeapManager(this, 8 << 20);
```

On higher resolution FB’s this hardcoded limit results in surfaceflinger allocation failures and the eventual restart of Android.
UI elements

- Assumes a certain set of peripherals
  - Telephony (3G signal indicator hardcoded)
  - Wifi (Wifi signal indicator hardcoded)
  - Ringer volume slider assumes telephony present
- Settings screen option assumes a handset
  - USB debugging option
  - SD card mount/unmount

```java
// phone_signal
mPhone = (TelephonyManager)context.getSystemService(Context.TELEPHONY_SERVICE);
mPhoneData = IconData.makeIcon("phone_signal",
   null, com.android.internal.R.drawable.stat_sys_signal_null, 0, 0);
mPhoneIcon = service.addIcon(mPhoneData, null);
service.setIconVisibility(mPhoneIcon, !hwNoPhone);
```
Hardcoded product policy

- Installation of non-marketplace .apks
  - Custom Android-based product may want this out-of-the-box instead of a settings option
- Enabling adb debugging
  - Many devices may want this enabled by default, except for a closed device
Dalvik VM internal structures

JValue is implemented in a LE specific way:

```c
typedef union JValue {
    u1 z;
    s1 b;
    u2 c;
    s2 s;
    s4 i;
    s8 j;
    float f;
    double d;
    void* l;
} JValue;
```

Requirement to access same value stored as byte as an integer

```c
JValue *jv = foo;
jv->b = 0x54;

print jv->i -> should output 0x54;
```

Key Character maps are LE

Prebuilt icu4c LE maps

Lots of missing htons/htonl use
Ethernet support

- Off the shelf Android doesn’t have good Ethernet support
- Early efforts just used a script to run the cmdline Android netcfg app to force dhcp configuration
- Requires registration of new connection type to manage link status and network available information similar to Wifi
- android-x86 project has a partially working Ethernet monitor
  - Problems with not always detecting link changes and re-dhcping
  - Doesn’t update Android-specific DNS properties
    - Yes, resolv.conf isn’t used in Android
Community issues

- Android Open Source Project (AOSP)
  - Relatively immature compared to traditional Linux communities
  - Huge lag in code being used by OHA member and what is dumped into the AOSP trees
  - Google developers generally don’t develop in the AOSP tree
  - Slowness in accepting code into the AOSP tree
  - OTOH, Google people on the AOSP lists are very responsive and helpful

- Alternative architectures (x86) are hosted at different sites
Pixelflinger JIT portability

- Codeflinger JIT designed with ARM opcodes in mind
- Other arches are able to be supported (MIPS/PPC), but it is significant work
Android layers its PM model on top of standard Linux PM
- Android wakelock concept
  - Applications can hold wakelocks to prevent system from sleeping
  - Once wakelocks are released cpu and peripherals may sleep

Android PM policy is hardcded to a handset model
- Full wakelock keeps CPU active and backlights at full brightness
- Partial wakelock allows bakclights to dim while CPU acive
- Modifying this policy for non handset designs requires modification of the PowerManager code.
Google provides lots of nice unit tests using the JUnit framework and a harness to execute them.

Unfortunately, many of them fail on the AOSP tree.

- Even on the emulator!

Goldfish results (1.5r1 release)
- passed: 67 test(s)
- failed: 4 test(s)
- failed: 044-proxy
- failed: 057-iteration-performance
- failed: 062-character-encodings
- failed: 071-dexfile
Google Apps

- Running Android on your device does not mean you can leverage the Marketplace
- Google’s App suite is proprietary software and use in devices is carefully controlled
  - Marketplace
  - Maps
  - CalendarProvider
- Cyanogen learned this in a widely publicized manner (http://android-developers.blogspot.com/2009/09/note-on-google-apps-for-android.html)
Android is different from traditional Linux
- When most people think of Linux, they think of a GNU/Linux distro
- Departure from accepted userspace components (HAL, udev, etc.)

Android has a lot of handset-focused policies hardcoded in the userspace code
- This is better than policy in kernel space

Solution is to continue to grow the AOSP community
- Community will need to unify
Questions