Portable Lunch/Scaling Problem

Mark.gross@intel.com

Android-Linux kernel Architect

February 2013

About me

I’ve been doing Android since 2008 and Linux kernel since 2001

What I do at Intel these days:

- Help to solve the embedded problem.
- Help to solve the scaling problem
- Help get ahead of anticipated transitions
- Help with SW bringing up on new architectures and devices
- Do code reviews
- Teach and help others
- Help with unplanned surprises

This talk gave me a scare

- While spelunking the Intel/psi tree for examples, many more components than anticipated seemed to need target device information at build time.
  - Is my thesis wrong?
  - I don’t think so.
    - Additional static analysis gives me confidence

Outline

Business-need discussion

High-level options

FUD / inertia

A little about Android build system

Static Analysis

Summary

Steps to take

Conclusions

We have too much differentiation ☺

We have 16 TARGET_DEVICEs

- We have 22 REV_DEVICE_NAMEs

Every year we get more lunch targets and more SOCs.

- Today we have 4 different SOCs, each having multiple lunch targets.
- Next year, I expect 2 or 3 more SOCs to show up.
- Next 2 years, I expect many more devices/targets.

We are limited

CI process can only handle O(100) changes a day worldwide.

Build times are long.

- Engineers testing on multiple targets spend too much time building targets.
- Buildbot servers are under stress.

Vendor/intel or device/intel is now hard to maintain.

- Adding new device targets is a pain.

Self-imposed Constraints

For all lunch targets we have:

- Shared repo
- Single branch

Branches used mostly for stable releases.

Linux kernel in our repo.

Test before merge process

Gerrit-based

For what?

15 foo.$(TARGET_DEVICE).so files (6MB)*

27 fw files (11MB)* that may or may not get used at runtime.

bzImage differs in .config used.

- This is getting solved ☺
- ONE bzImage for all targets in 2013!
- Go see Andrew Boie’s talk -- Tuesday @11:30!

init.rc files contain slight differences

* data from redhookbay-eng lunch target build output.

What can we do?

Note: FW and Kernel know target device we are running as well as SOC and FRU data.

- Lunch target that is binary compatible
- Uber-target: build all variants of all components
  - Use smart installer or runtime namespace manipulation at early boot.
- Build engine changes to enable reuse of portable intermediates.
  - create multiple root + system directories?
- More build servers
- Put 32-way build hosts on every developer’s desk

Why not?

A few people feel it’s impossible to fix this.

A few people feel it’s a dumb waste of time to fix this.

- Buy more servers! They are cheaper than developer time.

Inertia

Static Analysis

How many make targets use build time target data?

How many make targets use build time target data accessed through ro.product.*?

How many targets use platform data extracted from the kernel at runtime?

- How is this information used?
- Why is it needed?

Android Build System

Mega make

- Searches tree for Android*.mk files. (more or less)
- Important make variables that differ per target:
  - TARGET_DEVICE
  - REF_DEVICE_NAME ← Intel construct to deal with steppings
  - TARGET_BOARD_PLATFORM
  - PRODUCT_PACKAGES
  - BOARD_HAVE_*
  - BOARD_USES_*

Stats

Make VAR use	*.mk files using it non-trivially that matter
TARGET_DEVICE	14 (out of 143 files in entire tree)
REF_DEVICE_NAME	22
TARGET_BOARD_PLATFORM	9 (out of 63)
BOARD_HAVE_*	5
BOARD_USE_*	3

Make VAR definitions	Number of variants
TARGET_DEVICE	16
REF_DEVICE_NAME	10
TARGET_BOARD_PLATFORM	4
BOARD_HAVE_*	10
BOARD_USE_*	14

Android Runtime

Android has properties that communicate build time and device-specific data

- ro.product., build.props
- Build time props
  - ADDITIONAL_DEFAULT_PROPERTIES
  - ADDITIONAL_BUILD_PROPERTIES

Intel HW has platform data available at runtime

- SPID – soc + board fingerprint
  - Includes FRU information

There is also android.boot.* and some ro.* props

System/build.props that Matter

ro.product

- .model, .name, .device, .board, .platform

Used in 9 places:

- ./frameworks/base/core/java/com/android/internal/content/NativeLibraryHelper.java
- ./frameworks/base/core/jni/AndroidRuntime.cpp
- ./frameworks/base/wifi/java/android/net/wifi/WifiStateMachine.java
- ./frameworks/base/voip/jni/rtp/AudioGroup.cpp
- ./frameworks/av/media/mtp/MtpServer.cpp
- ./hardware/libhardware/hardware.c
- ./hardware/intel/libintelprov/droidboot.c
- ./external/webkit/Source/WebKit/android/jni/WebViewCore.cpp
- ./system/core/adb/adb.c

Stats

Build time props	Use count
build.props	10
ADDITIONAL_BUILD_PROPERITES	9
ADDITIONAL_DEFAULT_PROPERTIES	8
/sys/spid/*	9 (3 trivially used to compute file names)

Build time props	Number of variants
ADDITIONAL_BUILD_PROPERITES	14
ADDITIONAL_DEFAULT_PROPERTIES	12
/sys/spid/*	na

Where does all this leave us?

53 out of 3968 make files use target-specific info at build time in a non-trivial manner

18 out of 1491 runtime modules use target- specific info defined at build time at runtime.

9 components use build.props at runtime

9 modules use target-specific info at runtime gathered from kernel

* Data derived from the redhookbay-eng lunch target build

humph

Problem is still pretty complex.

Problem is bounded to a small set of files/modules compared to the full build.

Can we do this in steps?

Are there compromises that get us closer to a solution that would be good enough…for now?

recommendations

- Prefer use of build.prop data over compile time flags wherever possible.
    - Look into swapping out /system/build.prop file for correct target in early boot or at install time.
- When using runtime checks isn't enough, use unique LOCAL_MODULE names.
    - Enable all HAL module variants to be built in a single target.
- Scrub build for redundant or no longer used build time variables.
- Isolate where you define build time variables to easy-to-find and sensible files.
- Do we really need to use BOARD_USE and BOARD_HAS make variables?
    - Can we make them into build.props?

Summary

We have a need to scale to support large numbers of targets.

There are < 100 files that are causing trouble by using build time target data.

Still the problem is difficult.

- We should be able to make it work.
- Reduce number of build time flags.
- Prefer use of getprop(“ro.build….”) to compile time logic.
- Enable building all HAL modules in a single target.

We are just getting started

Questions?