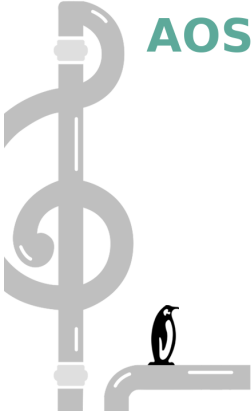

# AOSP: A case study in writing a custom build system

Linux Plumbers Build Systems MC, Vienna, 2024

Chris Simmonds

LINUX PLUMBERS CONFERENCE | Vienna, Austria
Sept. 18-20, 2024

# The AOSP build system

- AOSP build system is similar to others (OpenEmbedded, Buildroot), with its own peculiarities
- AOSP is a huge project (>50 MLOC of C++, Rust, Java, Kotlin, ...)
- The AOSP build system has evolved over time into something quite unique
- It does not include building a Linux kernel, bootloader, or any other ancillary binaries. It's up to the you (or the SoC vendor) to piece it all together (resulting in some truly weird stuff)

# Inputs

- Code written by Android developer team
- Third party code from many OSS projects in `$AOSP/externals/`
  - AOSP forks each one: there is no support for building from upstream code
  - the fork is mostly providing Android recipes to build the code, but may also include random changes
  - verson mapping is not clear
  - creates a drag on updating external code
- BSPs to support dev boards (from Linaro and Baylibre as far as I know) in `$AOSP/devices/`
- HALs and drivers from Qualcomm, Samsung and a few others in `$AOSP/hardware/`

# Outputs

- System images for target devices and emulators (Goldfish emulator based on QEMU, Cuttlefish emulator based on crosvm)
- Image files are typically in ext4, erofs, f2fs format
- SDKs that you can import into Android Studio

# Getting the code

- Each component is a git repository
- All repositories are downloaded at the start: no on-demand fetching
- The set of repositories needed is listed in a manifest
- The main Android manifest is at
  `https://android.googlesource.com/platform/manifest`
  - branch for each major release, tags for minor releases
- `repo init` downloads the manifest and supporting files

```
$ repo init -u https://android.googlesource.com/platform/manifest -b android-14.0.0_r37
```

- Then `repo sync` iterates through the manifest and clones or updates each of the git repos referenced in the manifest to local disk (in A14 there are about 1250 of them)

```
$ repo sync
```

# Extending the manifest; BSPs

- You can extend a manifest by putting snippets into `$AOSP/.repo/local_manifests`

For example to add a bsp for the "marvin" device, you could add a file named `$AOSP/.repo/local_manifests/marvin-default.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest>
  <remote name="github" fetch="https://github.com/csimmonds" />

  <project path="device/sirius" name="marvin" revision="android14" remote="github"/>
</manifest>
```

- But in practice, most SoC vendors give you a completely new manifest file with changes all over the AOSP codebase

# Notes and critique

- Declaring all the repositories in the manifest instead of in the recipes is a bit of a pain
- Generally means that you download far more than you actually need (e.g. BSPs, build tools)
  - you could go through the manifest and remove repos, but you have to know exactly which ones
  - maintenance burden across Android versions
  - not really worth the time and effort
- size of ithe download is a problem: > 150GB
- made worse because toolchains, kernels, binaries for QEMU, etc., are stored as precompiled binary objects (in `$AOSP/prebuilts/`) Each version change requires a complete new copy of the object. Half the download (c. 60GB) is binaries

# Evolution of the build system

- 2008 C/1.5: GNU Make: single Makefile composed at build time from fragments (*.mk) (ref: Recursive Make Considered Harmful) But, Make does not scale well: slow to start up, even if there is no work to do; no progress indication
- 2016 N/7: Kati and Ninja

  Kati implements the logic built into makefiles and outputs a dependency tree as a ninja manifest

  Ninja schedules jobs to reach the goal - showing progress
- 2017 O/8: soong

  Soong was intended as a replacement for makefiles and Kati. New format: Blueprint

  Progress of Soong has been slow: in T/13 there are still 1000's of makefile fragments
- 2023 U/14: Bazel

  Bazel was intended to be a solution that would unify the build into a single tool ... but the project was cancelled before it was finished

# Kernel, bootloader, firmware

- ... all ignored by AOSP build
- No support for building a kernel; there is a separate build environment for the kernel (GKI) using Bazel since A13.
- No support for building bootloaders (although there is much interaction between Android and the bootloader to implement A/B OTA, rollback, Verity, etc)
- No support for building disk images (i.e. no wic)
- OEMs and other 3rd parties often extend the build system to fill in these gaps, e.g. glodroid is a good example (https://glodroid.github.io/)
- But hacks to the build scripts reduces portability and increases maintenance

# Selecting and building a target product

- Set up the shell environment

```
$ source build/envsetup.sh
```

- Select the target using lunch (a shell function defined in envsetup.sh)

```
lunch <product>-<release>-<variant>
 e.g.
$ lunch aosp_cf_x86_64_phone-trunk_stable-userdebug

<product> = aosp_cf_x86_64_phone
<release> = trunk_stable
<variant> = userdebug
  possible values for <variant>: user | userdebug | eng
```

- Each target is defined by an `AndroidProduct.mk` e.g. `aosp_cf_x86_64_phone-userdebug` comes from `device/google/cuttlefish/AndroidProduct.mk`:

```
COMMON_LUNCH_CHOICES := aosp_cf_x86_64_phone-userdebug
```

- Sticky once selected

# Running a build

m, mm, and mmm build modules written in either Android.bp or Android.mk files

```
m or make          build all modules for a makefile target (default droid)
mm                 unconditionally build the module in the cwd
mmm dir1,dir2,...  unconditionally build modules in directory list
```

The `droid` target for `m` and `make` invoke all tasks needed to generate the final images and other artifacts

`mm` and `mmm` only build the Android.bp and Android.mk files listed

# Selecting modules for the build

- Each product lists the Android modules to build in Makefile variable `PRODUCT_PACKAGES`

```
PRODUCT_PACKAGES += CuttlefishService vsoc_input_service
```

- ... which you can dump using `get_build_var`:

```
$ get_build_var PRODUCT_PACKAGES
[...] sample_camera_extensions.xml CuttlefishService vsoc_input_service e2fsck [...]
```

- The final results are image files in `out/target/product/[device name]`

```
$ cd out/target/product/vsoc_x86_64
$ ls *.img
boot.img
ramdisk.img
super.img
system.img
vendor.img
[...]
```

- Typically you flash these to the device using fastboot

# Recipes

- Android modules are defined in recipes in one of two formats
- Android.bp
  - written in blueprint, introduced in O/8
  - T/13 has > 8000 Android.bp files
- Android.mk
  - written in Makefile format
  - deprecated, but still hanging around
  - in T/13 there are about 1000

# Developer workflow

- Start a temporary project with

```
$ repo start <some-name>
```

- make changes
- add your changes

```
$ git add .
$ git commit -m "..."
```

- Upload your changes:

```
$ repo upload
```

- View your change in Gerrit using the link from the repo upload, e.g. https://android-review.googlesource.com/c/platform/frameworks/native/+/1098432

Reference https://source.android.com/docs/setup/start

# Comparing AOSP to OpenEmbedded

There are many common concepts

| OpenEmbedded | AOSP |
|---|---|
| bb recipe | Android.bp and Android.mk |
| local.conf | AndroidProducts.mk |
| machine.conf | BoardConfig.mk |
| bitbake command | soong (soong_ui and soong_build) |
| bbclass | logic in `build/make/core` and in `build/soong` (soong module types) |

# Things AOSP does well

- lunch
  - nice to have a list of possible target machines and to select one
  - nice to have an easy way to select user/userdebug/eng build
  - (lunch is somewhat like pre selecting the MACHINE and image in OE. AOSP does not have an equivalent of DISTRO)
- repo and gerrit
- adb, logcat, fastboot

# Points for discussion

- Writing build systems is hard, there are many corner cases
- Could AOSP benefit from experiences of others (OpenEmbedded, Buildroot)?
- Could others benefit from things AOSP does?
- Could there be a forum for build system maintainers?
- Could there be a meta build system, meta meta data?