What differs the Android Open Source Project from other Linux distributions?

Sergio Prado
Toradex
$ WHOAMI

- Embedded software developer for more than 20 years.

- Team Lead at Toradex (https://www.toradex.com/).

- Consultant and trainer at Embedded Labworks (e-labworks.com/en).

- Contributor of some open source projects, including Buildroot, Yocto Project and the Linux kernel.

- Sometimes write technical stuff at embeddedbits.org.
OBJECTIVES

1. Android vs Linux distributions (user perspective).
2. Android vs Linux distributions (application developer perspective).
3. Android vs Linux distributions (distribution maintainer perspective).
4. Understand the architecture of the Android operating system.
LINUX SYSTEM ARCHITECTURE

GNU/Linux System

Hardware

Bootloader

Linux kernel

C library (glibc, eglibc, uclibc, etc)

Library

Application

Application
ANDROID ARCHITECTURE

Android Platform

- App
- App
- App
- App

Java framework (system services and API)

Native layer (libraries, daemons and tools)

Linux kernel (with some extra “features”)

Bootloader

Hardware
ANDROID ARCHITECTURE

Android applications

Android API

System Services

ART / Zygote

Java Libraries

Libs

Tools

Daemons

Init

HAL

JNI/Socket/Binder

System call

Linux Kernel

Android Platform
ANDROID ARCHITECTURE

Android applications

Android API

System Services

ART / Zygote

Libs  Tools  Daemons  Init  HAL

Java Libraries

Android Platform

Linux Kernel

JNI/Socket/Binder

System call

API

Binder
LINUX KERNEL FOR ANDROID

- The Linux kernel needs some extra features to run Android:
  - **binder**: IPC/RPC kernel-based mechanism.
  - **ashmem**: shared memory allocator.
  - **low memory killer**: manage low memory situations.

- Patches available in the kernel common repository:
  - [https://android.googlesource.com/kernel/common/](https://android.googlesource.com/kernel/common/)

- Currently, a mainline kernel has the minimal features required to boot an Android system.
BRANCHING MODEL

Android Common Kernel Branching Model

- **android-mainline**
  - 2020 (Android 11 = R)
    - New LTS (5.4)
  - 2021 (Android 12 = S)
    - New LTS (5.x)
  - 2022 (Android 13 = T)
    - New LTS (5.y)
  - 2023

- **android12-5.x**
- **android13-5.x**
- **android11-5.4**
- **android12-5.4**
- **android-4.19**
- **android-4.19-stable**
- **android-4.14**
- **android-4.14-stable**
- **android-4.9-q (stable) EOL:1/2023**
- **android-4.4-p (stable) EOL:2/2022**
KERNEL PATCHES

$ git clone https://android.googlesource.com/kernel/common kernel-common

$ git checkout remotes/origin/android11-5.4

$ git log --oneline | grep "ANDROID:" | less
5427f8b72fc0 ANDROID: GKI: update xiaomi symbol list
ecb88922f501 ANDROID: GKI: update Vivo symbol list
32b24237266 ANDROID: sysrq: add vendor hook for sysrq crash information
42e516f6b23b ANDROID: ABI: update allowed list for galaxy
de198b0f2d39 ANDROID: GKI: update Vivo symbol list

$ git log --oneline | grep "ANDROID:" | wc -l
1157
ANDROID ARCHITECTURE

Android applications

Android API

System Services

ART / Zygote

Java Libraries

Libs | Tools | Daemons | Init | HAL

Linux Kernel

Android Platform

API

Binder

JNI/Socket/Binder

System call
AOSP

* The Android platform is the user space part of an Android operating system, and it is implemented in the Android Open Source Project (AOSP).

* The AOSP is made of hundreds of repositories (780 in Android 11).
  https://android.googlesource.com/

* The source code is managed with common tools like repo and git.
  
  $ repo init -u https://android.googlesource.com/platform/manifest
  $ repo sync

* Large! (Android 11 is 100GB of source code plus 115GB after one build).
SOURCE CODE LISTING

$ ls
Android.bp  dalvik  libcore  read-snapshot.txt
dart       developers  libnativehelper  sdk
bionic      development  Makefile  system
bootable    device  out  test
dbootstrap.bash  external  packages  toolchain
dbuild   frameworks  pdk  tools
dcompatibility  hardware  platform_testing

core

kernel

prebuilts
COMMUNITY AND COLLABORATION

* Several discussion groups are available to communicate with the community and the developers:
  https://groups.google.com/d/forum/android-platform

* Anyone can contribute to the project via the Gerrit code review tool.
  https://android-review.googlesource.com

* The evolution of the project and the features that will be available in a future version of Android are controlled exclusively by Google.
LICENSING

× The vast majority of software components are under the permissive Apache and BSD licenses.

× Some software components are under GPL/LGPL licenses.

× Some Google applications’ source code are closed (Google Play, Gmail, Google Maps, YouTube, etc).
  × These applications are available in a package called Google Mobile Services (GMS), and to obtain them it is necessary to certify the device (ACP).
BUILD SYSTEM

- In the past, the Android build system was purely based on makefiles.
  - Instructions for compiling each software component were defined in Android.mk files.

- This build system had several shortcomings, including low performance in incremental builds.

- It was replaced in the latest versions of Android with the Soong build system.
  
  https://android.googlesource.com/platform/build/soong
The rules for compiling the software components are defined in Blueprint files (Android.bp), which have a syntax similar to JSON.

Blueprint files are processed by the Blueprint tool, which produces .ninja files with all the rules for processing Android software components. [https://opensource.google.com/projects/blueprint](https://opensource.google.com/projects/blueprint)

The .ninja files are then processed by the Ninja tool, which will compile all the software components to generate the Android images. [https://ninja-build.org/](https://ninja-build.org/)
Not all make files (Android.mk) were converted to Blueprint files (Android.bp) during the migration to this new build system.

For this reason, there is a tool called kati is responsible for converting Android.mk files to .ninja files.
https://github.com/google/kati

All Android.mk files should gradually be converted to Android.bp in the next Android releases, eliminating the need to use the kati tool in the Android build system.
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)

LOCAL_SRC_FILES = helloworld.c
LOCAL_MODULE = helloworld
LOCAL_MODULE_TAGS = optional

include $(BUILD_EXECUTABLE)
cc_binary {
    name: "helloworld",
    srcs: ["helloworld.c"],
    tags: ["optional"],
}
SOONG

- Android.bp
- Android.mk
- .ninja

Blueprint

Kati

Ninja

Executables, libraries and images (system, vendor, data, etc)
BUILDING ANDROID

$ source build/envsetup.sh

$ lunch aosp_x86_64-eng

$ make

$ cd out/target/product/generic_x86_64/ && ls *.img

cache.img   super_empty.img   vbmeta.img
dtb.img     super.img        vendor_boot-debug.img
encryptionkey.img  system.img    vendor_boot.img
ramdisk-debug.img   system-qemu.img
ramdisk.img   userdata.img
ramdisk-qemu.img  userdata-qemu.img
ROOTFS ORGANIZATION

- The rootfs organization on Linux systems is (mostly) standardized (e.g. Filesystem Hierarchy Standard).
  http://www.pathname.com/fhs/

- And Linux distributions try to conform to this standard:
  - Applications expect this format.
  - Make it easier the life of users and developers when they need to work with different Linux systems.

- But Android is an exception!
Where is /sbin, /usr, /lib, etc?
PARTITION LAYOUT (BEFORE ANDROID 10)
PARTITION LAYOUT (ANDROID 10+)

- **odm**
  - /app
  - /bin
  - ...

- **vendor**
  - /app
  - /bin
  - ...

- **product**
  - /app
  - /bin
  - ...

- **system**
  - /cache
  - /d
  - /data
  - /system
  - /etc
  - /mnt
  - /odm
  - /root
  - /sbin
  - /sdcard
  - /vendor
  - /product
  - /proc
  - /sys
  - /dev
  - /acct

- **cache**

- **userdata**
  - /anr
  - /app
  - /app-private
  - /backup
  - /dalvik-cache
  - /data
  - /dontpanic
  - /local
  - /misc
  - /property
  - ...

- **sdcard**

- **Kernel Virtual FS**
  - proc
  - sysfs
  - tmpfs
  - cgroup
REMOTE CONNECTION

✗ Usually the debugging process in embedded Linux systems takes place via the a serial port or JTAG interface (for low level debugging).

✗ However, mobile and consumer devices such as cell phones and tablets do not normally have these interfaces.

✗ That is why Google has developed ADB (Android Debug Bridge).

https://developer.android.com/studio/command-line/adb
ADB

User

Host

ADB Client

ADB Server

Target

adb

Android Gadget Driver
$ adb devices
List of devices attached
emulator-5554 device

$ adb pull /bin/cat
/bin/cat: 1 file pulled, 0 skipped. 8.2 MB/s (384688 bytes in 0.045s)

$ adb shell
generic_x86_64:/ #
NATIVE LAYER

Android applications

API

Android API

Binder

System Services

JNI/Socket/Binder

ART / Zygote

System call

Libs
Tools
Daemons
Init
HAL

Linux Kernel

Java Libraries
One of the main components of an operating system based on the Linux kernel is the C library.

The C library implements the operating system’s API, providing an interface for applications to access kernel services through system calls.

Several C libraries are available for Linux systems, including glibc, uclibc-ng and musl.

Android has its own C library, Bionic!
BIONIC

× At least three reasons motivated Google to implement its own C library: license, speed and size.

× The implementation of Bionic is simple, lightweight and released under the BSD license (source code in `bionic/`).

× It does not have full POSIX support, which can make it difficult to build native Linux applications for Android.
BUSYBOX 1.31.0 (libbb/missing_syscalls.c)

#include <sys/syscall.h>

pid_t getsid(pid_t pid)
{
    return syscall(__NR_getsid, pid);
}

int sethostname(const char *name, size_t len)
{
    return syscall(__NR_sethostname, name, len);
}

struct timex;

int adjtimex(struct timex *buf)
{
    return syscall(__NR_adjtimex, buf);
}

int pivot_root(const char *new_root, const char *put_old)
{
    return syscall(__NR_pivot_root, new_root, put_old);
}

...
BUSYBOX

- It is common to use BusyBox on embedded Linux devices.
- Busybox provides the (re)implementation of common tools and applications such as an init program, a shell and several utilities to manipulate and configure the system.
- But Android does not ship with BusyBox!
TOOLBOX AND TOYBOX

- Android uses Toolbox and Toybox, both released under a BSD license.

- Toolbox is a tool implemented by Google and its source code is available at system/core/toolbox/.

- Toybox is a tool implemented by the community (started by Rob Landley, BusyBox ex-maintainer) and its source code is available at external/toybox/.

- Because these tools have some limitations, it is common to install BusyBox on an Android device, especially during development.
addgroup, adduser, adjtimex, ar, arp, arping, ash, awk, basename, bbconfig, bbsh, brctl, bunzip2, busybox, bzcacat, bzip2, cal, cat, catv, chat, chattr, chcon, chgrp, chmod, chown, chpasswd, chpst, chroot, chrt, chvt, cksum, clear, cmp, comm, cp, cpio, crond, crontab, cryptpw, cttyhack, cut, date, dc, dd, deallocvt, delgroup, deluser, depmod, devfsd, df, dhcprelay, diff, dirname, dmesg, dnsd, dos2unix, dpkg, dpkg-deb, du, dumpkmap, dumpleases, e2fsck, echo, ed, egrep, eject, env, envdir, envuidgid, ether_wake, expand, expr, fakeidentd, false, fbset, fbsplash, fdflush, fdformat, fdisk, fetchmail, fgrep, find, findfs, fold, free, freeramdisk, fsck, fsck_minix, ftput, ftpput, fuser, getenforce, getopt, getsebool, getty, grep, gunzip, gzip, halt, hd, hdparm, head, hexdump, hostid, hostname, httpd, hush, hwclock, id, ifconfig, ifdown, ifenslave, ifup, inetd, init, inotifyd, insmod, install, ip, ipaddr, ipcalc, ipcrm, ipcs, iplink, iproute, iptunnel, kbd mode, kill, killall, killall5, klogd, lsh, last, length, less, linux32, linux64, linuxrc, ln, load_policy, loadfont, loadmap, logger, login, lorange, logread, losetup, lpd, lpq, lpr, ls, lsattr, lsmod, lzmacat, makedevs, man, matchpathcon, md5sum, mdev, mesg, microcom, mkdir, mke2fs, mkfifo, mkfs_minix, mkmod, mkswap, mktemp, modprobe, more, mount, mountpoint, msh, mt, mv, nameif, nc, netstat, nice, nmeter, nohup, nslookup, od, openvt, parse, passwd, patch, pgrep, pidof, ping, ping6, pipe_progress, pivot_root, pkill, poweroff, printf, printv, printf, ps, pscan, pwd, raidautorun, rdate, rdev, readahead, readlink, readprofile, realpath, reboot, renice, reset, resize, restorecon, rm, rmdir, rmmod, route, rpm, rpm2cpio, rtcwake, run_parts, runcon, runlevel, runsv, runsvdir, rx, script, sed, selinuxenabled, sendmail, seq, sestatus, setarch, setconsole, setenforce, setfiles, setfont, setkeycodes, setlogcons, setsebool, setsid, setuidgid, sh, shalsum, showkey, slattach, sleep, softlimit, sort, split, start_stop_daemon, stat, strings, sty, su, slogin, sum, sv, svlogd, swappoff, swapon, switch_root, sync, syslogd, tac, tail, tar, taskset, tcpsvd, tee, telnet, telnetd, test, tftp, tftpd, time, top, touch, tr, traceroute, true, tty, ttytype, tune2fs, udhcpc, udhcpd, udpsvd, umount, uname, uncompress, unexpand, uniq, unix2dos, unlzma, unzip, uptime, usleep, uudecode, uuencode, vconfig, vi, vlock, watch, watchdog, wc, wget, which, who, whoami, xargs, yes, zcat, zc
TOOLBOX

getprop modprobe setprop start stop toolbox
acpi base64 basename blkid blockdev cal cat chattr chcon chgrp chmod chown chroot chrt cksum clear cmp comm cp cpio cut date dd devmem df diff dirname dmesg dos2unix du echo egrep env expand expr fallocate false fgrep file find flock fmt free freeramdisk fsfreeze fsync getconf getenforce getfattr getopt grep groups gunzip gzip head help hostname hwclock i2detect i2cdump i2cget i2cset iconv id ifconfig inotfyd insmod install ionice iorename iostat iotop kill killall ln load_policy log logname lsof lsattr lsmod lsof lspci lsusb makedevs md5sum microcom mkdir mkfifo mkod mkswap mktemp modinfo modprobe more mount mountpoint mv nbd-client nc netcat netstat nice nl nohup nproc nsenter od partprobe paste patch pgrep pidof ping ping6 pivot_root pkill pmap printenv printf prlimit ps pwd pwdx readelf readlink realpath renice restorecon rev rfkill rm rmdir rmmod runcon sed sendevent seq setenforce setfattr setsid sha1sum sha224sum sha256sum sha384sum sha512sum sleep sort split stat strings stty swapoff swapon sync sysctl tac tail tar taskset tee test time timeout top touch tr traceroute traceroute6 true truncate tty tunctl ulimit umount uname uniq unix2dos unlink unshare uptime usleep uudecode uuencode uuidgen vconfig vi vmstat watch wc which whoami xargs xxd yes zcat
INIT SYSTEM

- The init application is run by the kernel right after mounting the rootfs, and it is responsible for system initialization and management.

- There are several implementations of the init process for Linux systems, including sysvinit, systemd and upstart.

- As you may already expect, Android has its own init system!
The Android init process has 3 main responsibilities:

- Boot the operating system (export environment variables, create and set permissions on files and directories, create links, mount file systems, setup selinux, etc).
- Start and monitor daemons.
- Manage system properties.

The behavior of the init process is defined in a configuration file (by default /etc/init/hw/init.rc).
INIT SOURCE CODE (init.cpp)

```cpp
static void LoadBootScripts(ActionManager& action_manager, ServiceList& service_list) {
    Parser parser = CreateParser(action_manager, service_list);

    std::string bootscript = GetProperty("ro.boot.init_rc", "");
    if (bootscript.empty()) {
        parser.ParseConfig("/system/etc/init/hw/init.rc");
        if (!parser.ParseConfig("/system/etc/init")) {
            late_import_paths.emplace_back("/system/etc/init");
        }
        // late_import is available only in Q and earlier release. As we don't
        // have system_ext in those versions, skip late_import for system_ext.
        parser.ParseConfig("/system_ext/etc/init");
        if (!parser.ParseConfig("/product/etc/init")) {
            late_import_paths.emplace_back("/product/etc/init");
        }
        if (!parser.ParseConfig("/odm/etc/init")) {
            late_import_paths.emplace_back("/odm/etc/init");
        }
        if (!parser.ParseConfig("/vendor/etc/init")) {
            late_import_paths.emplace_back("/vendor/etc/init");
        }
    } else {
        parser.ParseConfig(bootscript);
    }
}
```
import /init.environ.rc
import /system/etc/init/hw/init.usb.rc
import /init.${ro.hardware}.rc
import /vendor/etc/init/hw/init.${ro.hardware}.rc
import /system/etc/init/hw/init.usb.configfs.rc
import /system/etc/init/hw/init.${ro.zygote}.rc

# Cgroups are mounted right before early-init using list from /etc/cgroups.json
on early-init
    # Disable sysrq from keyboard
    write /proc/sys/kernel/sysrq 0

    # Android doesn't need kernel module autoloading, and it causes SELinux
    # denials. So disable it by setting modprobe to the empty string. Note: to
    # explicitly set a sysctl to an empty string, a trailing newline is needed.
    write /proc/sys/kernel/modprobe \\

...
on init
    sysclktz 0

    # Mix device-specific information into the entropy pool
    copy /proc/cmdline /dev/urandom
    copy /system/etc/prop.default /dev/urandom

    symlink /proc/self/fd/0 /dev/stdin
    symlink /proc/self/fd/1 /dev/stdout
    symlink /proc/self/fd/2 /dev/stderr

    ...

# Mount filesystems and start core system services.
on late-init
    trigger early-fs

    # Mount fstab in init.{$device}.rc by mount_all command. Optional parameter
    # '--early' can be specified to skip entries with 'latemount'.
    # /system and /vendor must be mounted by the end of the fs stage,
    # while /data is optional.
    trigger fs
INIT.RC

on property:ro.debuggable=1
    # Give writes to anyone for the trace folder on debug builds.
    # The folder is used to store method traces.
    chmod 0773 /data/misc/trace
    # Give reads to anyone for the window trace folder on debug builds.
    chmod 0775 /data/misc/wmtrace

service ueventd /system/bin/ueventd
    class core
    critical
    seclabel u:r:ueventd:s0
    shutdown critical

service console /system/bin/sh
    class core
    console
    disabled
    user shell
    group shell log readproc
    seclabel u:r:shell:s0
    setenv HOSTNAME console
Android currently uses MirBSD Korn Shell.
https://www.mirbsd.org/mksh.htm

Probably much more limited than the default shell available in your Linux desktop.

That means some scripts that run in your desktop may not run on Android!
Daemons are processes that run in the background and are responsible for managing some system functionality:

- Most of them are executed at startup by the init process.
- Usually they run in the background for as long as the system is functional.
- They are normally used to control and centralize access to a system resource.

On Android, many daemons are an interface between the Android framework and system resources (network, storage, energy, etc).
ANDROID DAEMONS

✦ **ueventd**: responsible for managing the connection of hardware devices (device hotplugging). It is the equivalent of udev or mdev on Linux systems.

✦ **vold**: (volume daemon) is responsible for monitoring events from storage devices.

✦ **ril**: (radio interface layer daemon) manages communication with the modem chip (voice and data).
ANDROID DAEMONS

- **netd**: (network management service daemon) is responsible for managing network connections (Bluetooth, Wi-Fi, USB, etc).

- **installd**: (install daemon) is responsible for managing the installation of Android applications (* .apk) and their associated resources.

- **lmkd**: (low memory killer daemon) is responsible to manage the kernel low memory killer interface.
<table>
<thead>
<tr>
<th>USER</th>
<th>PID</th>
<th>PPID</th>
<th>VSZ</th>
<th>RSS</th>
<th>WCHAN</th>
<th>ADDR S NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>1</td>
<td>0</td>
<td>10782796</td>
<td>9696</td>
<td>do_epoll_+</td>
<td>0 S init</td>
</tr>
<tr>
<td>root</td>
<td>122</td>
<td>1</td>
<td>10761204</td>
<td>7376</td>
<td>do_sys_po+</td>
<td>0 S ueventd</td>
</tr>
<tr>
<td>logd</td>
<td>145</td>
<td>1</td>
<td>10764228</td>
<td>7932</td>
<td>__x64_sys+</td>
<td>0 S logd</td>
</tr>
<tr>
<td>lmkd</td>
<td>146</td>
<td>1</td>
<td>10756496</td>
<td>2456</td>
<td>do_epoll_+</td>
<td>0 S lmkd</td>
</tr>
<tr>
<td>system</td>
<td>147</td>
<td>1</td>
<td>10759476</td>
<td>5616</td>
<td>do_epoll_+</td>
<td>0 S servicemanager</td>
</tr>
<tr>
<td>system</td>
<td>148</td>
<td>1</td>
<td>10761244</td>
<td>6498</td>
<td>do_epoll_+</td>
<td>0 S hwservicemanager</td>
</tr>
<tr>
<td>system</td>
<td>149</td>
<td>1</td>
<td>10759572</td>
<td>4828</td>
<td>do_epoll_+</td>
<td>0 S vndservicemanager</td>
</tr>
<tr>
<td>root</td>
<td>153</td>
<td>1</td>
<td>10770896</td>
<td>8732</td>
<td>binder_th+</td>
<td>0 S vold</td>
</tr>
<tr>
<td>tombstoned</td>
<td>250</td>
<td>1</td>
<td>10755388</td>
<td>2128</td>
<td>do_epoll_+</td>
<td>0 S tombstoned</td>
</tr>
<tr>
<td>statsd</td>
<td>266</td>
<td>1</td>
<td>10766140</td>
<td>4572</td>
<td>do_epoll_+</td>
<td>0 S statsd</td>
</tr>
<tr>
<td>root</td>
<td>267</td>
<td>1</td>
<td>10781776</td>
<td>9532</td>
<td>binder_th+</td>
<td>0 S netd</td>
</tr>
<tr>
<td>credstore</td>
<td>306</td>
<td>1</td>
<td>10764440</td>
<td>7296</td>
<td>binder_th+</td>
<td>0 S credstore</td>
</tr>
<tr>
<td>gpu_service</td>
<td>307</td>
<td>1</td>
<td>10762672</td>
<td>6804</td>
<td>binder_th+</td>
<td>0 S gpuservice</td>
</tr>
<tr>
<td>system</td>
<td>308</td>
<td>1</td>
<td>10873496</td>
<td>31972</td>
<td>do_epoll_+</td>
<td>0 S surfaceflinger</td>
</tr>
<tr>
<td>root</td>
<td>316</td>
<td>1</td>
<td>10756876</td>
<td>2656</td>
<td>do_sys_po+</td>
<td>0 S netmgr</td>
</tr>
<tr>
<td>root</td>
<td>318</td>
<td>1</td>
<td>10758880</td>
<td>3072</td>
<td>do_sys_po+</td>
<td>0 S wifi_forwarder</td>
</tr>
<tr>
<td>wifi</td>
<td>320</td>
<td>1</td>
<td>10759960</td>
<td>5464</td>
<td>do_select</td>
<td>0 S hostapd_nohidl</td>
</tr>
<tr>
<td>logd</td>
<td>326</td>
<td>1</td>
<td>10756544</td>
<td>3160</td>
<td>__skb_wai+</td>
<td>0 S logcat</td>
</tr>
<tr>
<td>root</td>
<td>352</td>
<td>1</td>
<td>10773084</td>
<td>6376</td>
<td>0</td>
<td>0 S adbd</td>
</tr>
<tr>
<td>nobody</td>
<td>354</td>
<td>1</td>
<td>10757496</td>
<td>3164</td>
<td>do_sys_po+</td>
<td>0 S traced_probes</td>
</tr>
<tr>
<td>nobody</td>
<td>355</td>
<td>1</td>
<td>10757632</td>
<td>3464</td>
<td>do_sys_po+</td>
<td>0 S traced</td>
</tr>
<tr>
<td>camerarserver</td>
<td>356</td>
<td>1</td>
<td>58984</td>
<td>17248</td>
<td>binder_th+</td>
<td>0 S camerarserver</td>
</tr>
<tr>
<td>drm</td>
<td>357</td>
<td>1</td>
<td>25952</td>
<td>6512</td>
<td>binder_th+</td>
<td>0 S drmserver</td>
</tr>
<tr>
<td>incidentd</td>
<td>359</td>
<td>1</td>
<td>10761968</td>
<td>4992</td>
<td>do_epoll_+</td>
<td>0 S incidentd</td>
</tr>
<tr>
<td>root</td>
<td>360</td>
<td>1</td>
<td>10765704</td>
<td>6452</td>
<td>binder_th+</td>
<td>0 S installd</td>
</tr>
<tr>
<td>iorapd</td>
<td>361</td>
<td>1</td>
<td>10775424</td>
<td>9536</td>
<td>futex_wai+</td>
<td>0 S iorapd</td>
</tr>
<tr>
<td>keystore</td>
<td>362</td>
<td>1</td>
<td>10764916</td>
<td>7404</td>
<td>binder_th+</td>
<td>0 S keystore</td>
</tr>
<tr>
<td>root</td>
<td>366</td>
<td>1</td>
<td>10765596</td>
<td>5648</td>
<td>binder_th+</td>
<td>0 S storaged</td>
</tr>
</tbody>
</table>

ANDROD DAOEMONS
LOGGING

✓ The log daemon (logd) is responsible for managing logs in Android.

✓ Access to the logs is done through sockets exported in /dev/socket/.

  # ls /dev/socket/logd*
  /dev/socket/logd  /dev/socket/logdr  /dev/socket/logdw

✓ To read or write to the logs, it is not necessary to directly access these sockets. For this, applications can use the liblog library.

✓ In the terminal, the user can write to the log with the log command and read/control the logs through the logcat tool.
10-14 13:36:51.722   771   934 D SmsNumberUtils: enter filterDestAddr. destAddr="[BajqU4K5_YhSYbs-7QUn0dOwcmI]"
10-14 13:36:51.723   771   934 D SmsNumberUtils: destAddr is not formatted.
10-14 13:36:51.723   771   934 D SmsNumberUtils: leave filterDestAddr, new destAddr="[BajqU4K5_YhSYbs-7QUn0dOwcmI]"
10-14 13:36:57.054   316   316 E netmgr  : qemu_pipe_open_ns:62: Could not connect to the 'pipe:qemud:network' service:
10-14 13:36:57.054   316   316 E netmgr  : Failed to open QEMU pipe 'qemud:network': Invalid argument
10-14 13:36:57.325   318   318 E wifi_forwarder: RemoteConnection failed to initialize: RemoteConnection failed to open pipe
10-14 14:37:45.408   494  1324 D WifiNl80211Manager: Scan result ready event
10-14 14:37:45.408   494  1324 D WifiNative: Scan result ready event
10-14 14:37:59.109   316   316 E netmgr : Failed to open QEMU pipe 'qemud:network': Invalid argument
10-14 14:37:59.574   318   318 E wifi_forwarder: qemu_pipe_open_ns:62: Could not connect to the 'pipe:qemud:wififorward' service:
10-14 14:37:59.575   318   318 E wifi_forwarder: RemoteConnection failed to initialize: RemoteConnection failed to open pipe
10-14 14:38:00.003   642   642 D KeyguardClockSwitch: Updating clock: 2:38
10-14 14:38:00.003   642   642 D KeyguardClockSwitch: Updating clock: 2:39
10-14 14:38:59.142   316   316 E netmgr : qemu_pipe_open_ns:62: Could not connect to the 'pipe:qemud:network' service:
10-14 14:38:59.142   316   316 E netmgr : Failed to open QEMU pipe 'qemud:network': Invalid argument
10-14 14:39:00.003   642   642 D KeyguardClockSwitch: Updating clock: 2:40
HARDWARE ABSTRACTION LAYER

* On an embedded Linux system, access to hardware devices is exposed to applications via entries in /dev or /sys.

* Android relies on an additional layer (Hardware Abstraction Layer) to abstract access to hardware devices.

* Some motivations for this abstraction layer:
  * Decouple the hardware access from the Android framework.
  * Freedom for the manufacturer to implement the hardware access logic in the HAL code and release under any software license.
HAL ON EMBEDDED LINUX
HAL ON ANDROID

Applications

API (android.*)

System Services

HAL

Driver

Linux Kernel

Kernel space

User space

C/C++

Java
EXAMPLE: SERIAL PORT

Kernel Linux

Driver UART

Serial Port HAL

Serial Port Service

Serial Port Manager

Serial Port App

System call

/dev/ttyS0

Binder

Binder

API
BINDER

Framework

Hardware Service Manager
(hwservicemanager)

HALs

Binder driver (/dev/hwbinder)
FRAMEWORK LAYER

Linux Kernel

ART / Zygote

System Services

Android API

Java Libraries

API

JNI/Sockets/Binder

System call

Libs

Tools

Daemons

Init

HAL

Linux Kernel

Init

HAL

Daemons

Libs

Tools

JNI/Sockets/Binder

System call

API

ART / Zygote

System Services

Android API

Java Libraries

FRAMEWORK LAYER
# service list

Found 184 services:

0   DockObserver: []
1   SurfaceFlinger: [android.ui.ISurfaceComposer]
2   accessibility: [android.view.accessibility.IAccessibilityManager]
3   account: [android.accounts.IAccountManager]
4   activity: [android.app.IActivityManager]
5   activity_task: [android.app.IActivityTaskManager]
6   adb: [android.debug.IAdbManager]
7   alarm: [android.app.IAlarmManager]
8   android.hardware.identity.IIdentityCredentialStore/default: [android.hardware.identity.IIdentityCredentialStore]
9   android.hardware.light.ILights/default: [android.hardware.light.ILights]
10  android.hardware.power.IPower/default: [android.hardware.power.IPower]
11  android.hardware.rebootscreen.IRebootEscrow/default: [android.hardware.rebootscreen.IRebootEscrow]
12  android.hardware.vibrator.IVibrator/default: [android.hardware.vibrator.IVibrator]
13  android.security.identity: [android.security.identity.ICredentialStoreFactory]
14  android.security.keystore: [android.security.keystore.IKeystoreService]
15  android.service.gatekeeper.IGateKeeperService: [android.service.gatekeeper.IGateKeeperService]
16  app_binding: []
17  app_integrity: [android.content.integrity.IAppIntegrityManager]
18  appops: [com.android.internal.app.IAppOpsService]
19  appwidget: [com.android.internal.appwidget.IAppWidgetService]
20  audio: [android.media.IAudioService]
21  auth: [android.hardware.biometrics.IAuthService]
...

SERVICES ARCHITECTURE

- **Kernel**
  - **Daemon/HAL**
  - **System Service**
  - **Android API**
  - **Android App**

- **API call**
- **Binder**
- **Binder/Socket/JNI**
- **System call**

- **Abstracts the access**
- **Centralizes the access**
- **Checks permission**
CALLING SERVICES FROM COMMAND LINE!

# service call
service: No code specified for call
Usage: service [-h|-?]
   service list
   service check SERVICE
   service call SERVICE CODE [i32 N | i64 N | f N | d N | s16 STR | null | fd f | nfd n | afd f ] ... 
Options:
   i32: Write the 32-bit integer N into the send parcel.
   i64: Write the 64-bit integer N into the send parcel.
   f: Write the 32-bit single-precision number N into the send parcel.
   d: Write the 64-bit double-precision number N into the send parcel.
   s16: Write the UTF-16 string STR into the send parcel.
   null: Write a null binder into the send parcel.
   fd: Write a file descriptor for the file f to the send parcel.
   nfd: Write file descriptor n to the send parcel.
   afd: Write an ashmem file descriptor for a region containing the data from file f to the send parcel.

# service call phone 2 s16 com.android.launcher3 s16 12345678
Result: Parcel(00000000    '....')
APPLICATION LAYER

API

Binder

JNI/Sockets/Binder

System call

Applications

Android API

System Services

ART / Zygote

Java Libraries

Libs

Tools

Daemons

Init

HAL

Linux Kernel
ANDROID APPLICATIONS

* Well defined set of APIs.

* Android applications are written Java or Kotlin using the Google SDK.

* They are packaged in files with the .apk extension, that contains the compiled code, data and resources used by the application.

* They can be installed via Google Play or manually via ADB or any file manager.
APPLICATION COMPONENTS

✗ Android applications are basically composed of 4 types of components:
  ✗ Activities.
  ✗ Services.
  ✗ Broadcast receivers.
  ✗ Content providers.
APPLICATION COMMUNICATION

✗ An application can contain one or more components.

✗ An application can start any component, including components from other applications.

✗ The components communicate with each other through a message exchange mechanism called **intent**.
INTENT
WHAT ABOUT FRAGMENTATION?
CONCLUSION

✗ Similarities: Based on the Linux kernel.

✗ Differences: (almost) everything else!

✗ So is Android a Linux distribution or not?
REFERENCES

- Android official platform documentation: https://source.android.com/
- Android source code! https://android.googlesource.com/
- Karim Yaghmour (Opersys) talks: https://www.youtube.com/channel/UCWIZcsPiXb9fQWJcijUtFXg
Q&A

Sergio Prado
sergio@embeddedbits.org

https://twitter.com/sergioprado
https://www.linkedin.com/in/sprado

Thank you!