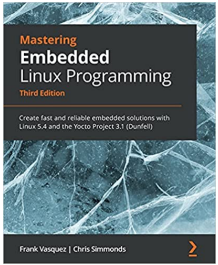
Exploring Android internals with ADB

Chris Simmonds

Droidcon London 2022

License

About Chris Simmonds



- Consultant and trainer
- Author of *Mastering Embedded Linux Programming*
- Working with embedded Linux since 1999
- Android since 2009
- Speaker at many conferences and workshops

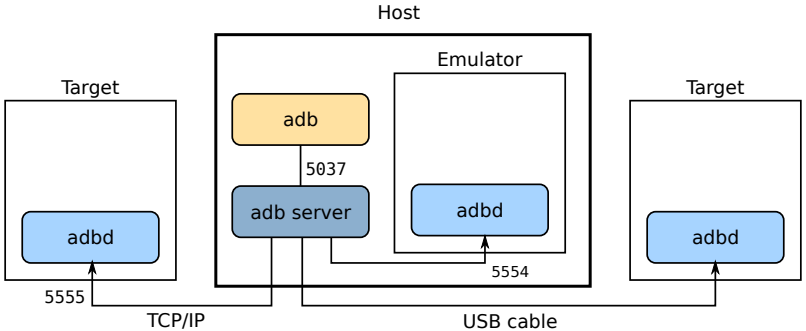
"Looking after the Inner Penguin" blog at `https://2net.co.uk/`

@2net_software

`https://uk.linkedin.com/in/chrisdsimmonds/`

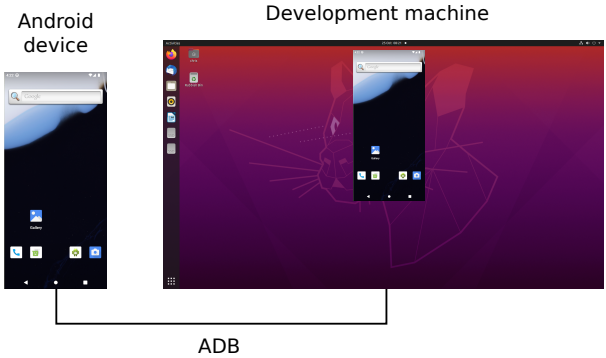
- Introduction
- System Services
- Native Services
- Sandboxing
- System properties

Tools: ADB



ADB is the link between development machine and Android device

Tools: scrcpy

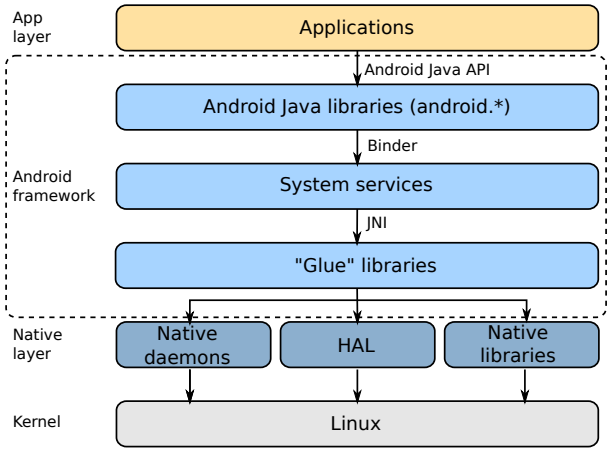


scrcpy = screen copy: remote display and touch input, does not require root

https://github.com/Genymobile/scrcpy

What is Android?

Architecture of Android



- Introduction
- System Services
- Native Services
- Sandboxing
- System properties

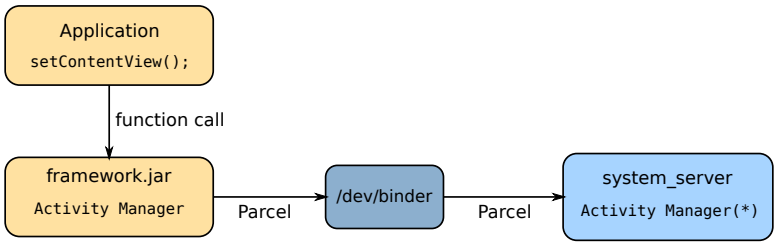
System services

- The Android framework is an object-oriented operating system implemented on top of a conventional POSIX operating system, Linux
- Implemented as a collection of **system services**
- System services have high privilege levels and access to all framework APIs
- Client programs (Android apps) send objects to system services to do low level tasks
- We can see system services from ADB using the `service` command

```
service list
service call <service name> <function number>
```

Binder

- **Binder** is the Inter Process Communication (IPC) used to communicate between Applications and Framework objects
- Messages are encoded into **parcels** that are sent from a **manager** interface to a **service**



(*) Should have been called 'Activity Service'

Demo time

System services

Package manager

- **Package manager** is a system service responsible for tracking packages and some system-wide attributes
  - installing and uninstalling packages
  - tracking permissions granted to packages
  - system features
  - platform libraries
- Command-line tools

```
dumpsys package
cmd package
pm
```

Packages

- The main database for Package Manager is `/data/system/packages.xml`

Dump entire package database

```
dumpsys package packages
```

List packages

```
pm list packages
```

Activity Manager

- **Activity Manager** is the system service that handles lifecycle events
- ... the scheduler for Android applications
- Command line tools:

```
dumpsys activity
logcat -b event
am
```

- Introduction
- System Services
- Native Services
- Sandboxing
- System properties

Native services

- Native services run lower level tasks, between framework and Linux
- Started by init
- boot sequence: power on – bootloader – linux – init – Android
- some important daemons:

adbd	ADB daemon
logd	Android log daemon
zygote	Parent process of Android Run Time
ueventd	creates device nodes in `/dev`
bootanim	shows the boot animation
lmkd	low memory killer daemon
vold	volume daemon, mounts external storage e.g. SD cards

Starting native daemons

- `/system/bin/init` is started by Linux at boot time
- Init parses "run command" files (.rc), starting with `/system/etc/init/hw/init.rc`
- There are other .rc files in `/system/etc/init` and `/vendor/etc/init`
  - about 100 in all
- The status of each daemon is recorded in property `init.svc.<daemon name>`

```
# getprop | grep init.svc
[init.svc.adbd]: [running]
[init.svc.audioserver]: [running]
[init.svc.bootanim]: [stopped]
[...]
init.svc.vendor.audio-hal]: [running]
[...]
```

Zygote

- zygote is a native daemon that launches ART and so is able to run DEX code
- Launches `system_server`, which starts Android
- Launches Android apps on demand
- Started by `/system/etc/init/hw/init.zygote64.rc`

Demo time

Native services

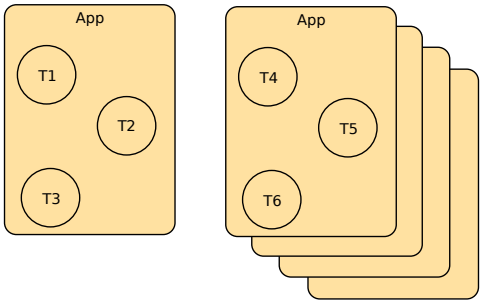
- Introduction
- System Services
- Native Services
- Sandboxing
- System properties

The application sandbox

- Application sandbox limits the memory and files that an application can see
- Android uses Linux processes for memory separation
- Android uses Linux User IDs (UID), Group IDs and file mode for file separation
  - also known as DAC, Discretionary Access Control

Memory separation

- A Linux process runs in a unique address space
  - threads in one app cannot read or write memory from another app



File separation

- Each App is assigned a unique UID by package manager
  - Linux user id == Android application ID
- Each app has a place to put private files: `/data/data/<package name>`

User ID

```
dumpsys package packages
[...]
  Package [com.android.camera2] (c3f65c):
    userId=10080
[...]
```

File permissions

```
ls -l /data/data/com.android.camera2/
total 24
drwxrws--x 3 u0_a80 u0_a80_cache 4096 2022-10-25 13:54 cache
drwxrws--x 2 u0_a80 u0_a80_cache 4096 2022-10-25 12:22 code_cache
drwxrwx--x 2 u0_a80 u0_a80       4096 2022-10-25 13:54 shared_prefs
```

SELinux

- Another layer of security
- Each process has an SELinux context, shown with `ps -Z`:

```
ps -AZ
u:r:platform_app:s0:c512,c768      u0_a98     753   373 14257512 225728 do_epoll_wait      0 S com.ar
u:r:priv_app:s0:c512,c768          u0_a91    1090   373 13993028 153572 do_epoll_wait      0 S com.ar
u:r:system_app:s0                  system    1890   373 13838600 86012 do_epoll_wait       0 S com.ar
u:r:untrusted_app_25:s0:c512,c768 u0_a86     2282   373 13890380 143056 do_epoll_wait      0 S com.ar
```

- Each file also has an SELinux context `ls -Z`:

```
# ls -Z /data/data/com.android.camera2
u:object_r:app_data_file:s0:c80,c256,c512,c768 cache
u:object_r:app_data_file:s0:c80,c256,c512,c768 code_cache
u:object_r:app_data_file:s0:c80,c256,c512,c768 shared_prefs
```

Demo time

Processes, UIDs, GIDs, and file permissions

More about the ADB shell

- **user** build: always `shell`

```
$ id
uid=2000(shell) gid=2000(shell) [...]
```

- **userdebug** build: start as `shell`, but can switch user (su) to root

```
$ id
uid=2000(shell) gid=2000(shell) [...]
$ su
# id
uid=0(root) gid=0(root)
```

- **eng** build: always root

```
# id
uid=0(root) gid=0(root)
```

Command-line tools

- Most of the command line tools are implemented in **toybox**
- `/system/bin/toybox` is a multi-call binary that implements about 200 utilities, type `toybox` to get the list

```
$ toybox
[ acpi base64 basename blkdiscard 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 i2cdetect
i2cdump i2cget i2cset iconv id ifconfig inotifyd insmod install ionice iorenice iotop kill
killall ln load_policy log logname losetup ls lsattr lsmod lsof lspci lsusb makedevs md5sum
microcom mkdir mkfifo mknod 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 rtcwake 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 uclampset ulimit umount uname uniq unix2dos unlink unshare uptime
usleep uudecode uuencode uuidgen vconfig vi vmstat watch wc which whoami xargs xxd yes zcat
```

- Introduction
- System Services
- Native Services
- Sandboxing
- System properties

System properties

- **System properties** are a global store of name/value pairs, used for:

Build information

```
getprop ro.build.version.release
13
```

Status

```
getprop init.svc.adbd
[init.svc.adbd]: [running]
```

Configuration set at build time

```
getprop ro.sf.lcd_density
420
```

Local configuration

```
getprop persist.sys.timezone
Europe/London
```

Where next?

- The best way to learn about Android is to build it and modify it
- The Android operating system is open source, available via AOSP (Android Open Source Project)
- Instructions about building here(*):
  `https://source.android.com/docs/setup/start/initializing`
- AOSP community
  - The AOSP and AAOS Meetup: `https://aospandaaos.github.io/`
  - AOSP Developers Community
    `https://aosp-developers-community.github.io/`

OK, building AOSP requires lots of hardware (32 GB RAM, 200 GB disk, 8 or more cores) and time (many hours), but it's still worth it

Questions?

Slides: https://2net.co.uk/slides/android-internals-csimmonds-droidcon-london-2022.pdf

Training: https://2net.co.uk/training/embedded-android

@2net_software

https://uk.linkedin.com/in/chrisdsimmonds/