The Android Concept Journey

Version 1.0
May-2025

Created using Craiyon, AI Image Generator

Introduction.....3
AOSP (Android Open Source Project).....4
OHA (Open Handset Alliance).....5
Android Auto.....6
Android Automotive (AAOS).....7
API Level (Application Programming Interface Level).....8
Android Emulator.....9
AVD (Android Virtual Device).....10
Activity.....11
Back Stack.....12
Application Sandbox.....13
Binder.....14

Introduction

When starting to learn Android I believe that they are basic concepts that everyone needs to know about. Because of that I have decided to write a series of short writeups aimed at providing a basic explanation for fundamental concepts which are part of the Android operating system.

Overall, I wanted to create something that will improve the overall knowledge of Android in writeups that can be read in 1-3 mins. I hope you are going to enjoy the ride.

Lastly, you can follow me on twitter - @boutnaru (https://twitter.com/boutnaru). Also, you can read my other writeups on medium - https://medium.com/@boutnaru. Lastly, You can find my free eBooks at https://TheLearningJourneyEbooks.com.

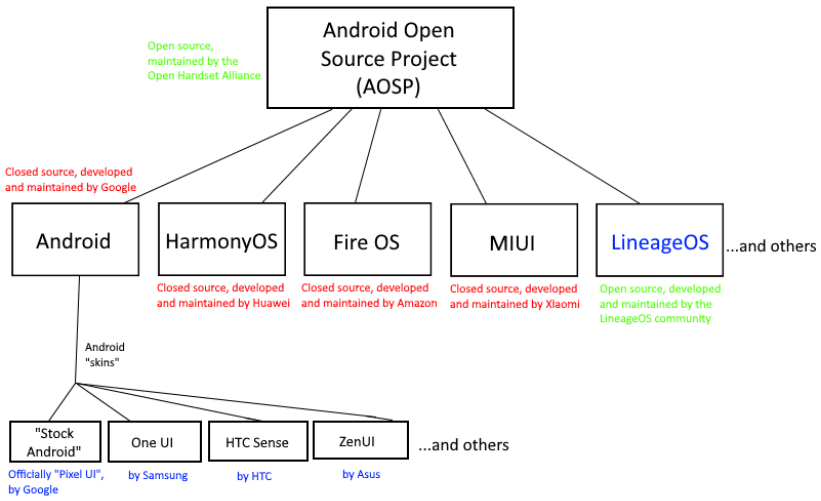
Lets GO!!!!!!

AOSP (Android Open Source Project)

AOSP (Android Open Source Project) is the repository of source code and the foundation which maintains it. ASOP is responsible for the core of the Android operating system. Thus, by leveraging AOSP everyone can download and create their own OS based on Android[1]. Examples for such cases are: Amazon's Fire OS[2] and LineageOS[3] - as shown in the diagram below[4].

Overall, the source code of the Android operating system is managed using a collection of Git repositories hosted by Google. Thus, we can use the repo client for downloading ("repo sync -c -j16", where 16 is number of threads for faster completion) the Android source code[5]. Since March 2025 the latest release branch will always be referenced by the new "android-latest-release" manifest, which can be used directly with repo[6].

Lastly, we can use the "Android Code Search" portal for searching for code and files across the entire source code[7]. Although, Android is free and open source (mostly licensed under the Apache License) most devices are running a proprietary version of Android and with closed software preinstalled[8].



[1] https://www.techtarget.com/searchmobilecomputing/definition/Android-Open-Source-Project-AOSP
[2] https://developer.amazon.com/docs/fire-tv/fire-os-overview.html
[3] https://lineageos.org/
[4] https://telegra.ph/Android-open-source-project-08-14
[5] https://source.android.com/docs/setup/download#initialize_the_repo_client
[6] https://source.android.com/docs/whatsnew/site-updates#aosp-changes
[7] https://cs.android.com/android/platform/superproject/main
[8] https://en.wikipedia.org/wiki/Android_(operating_system)

OHA (Open Handset Alliance)

OHA (Open Handset Alliance) is a group of 84 technology and mobile companies who have come together to accelerate innovation in mobile. Their goal is to offer consumers a richer, less expensive, and better mobile experience. This organization is behind the development of the Android operating system[9].

Overall, OTH was established in 2007 and is led by Google. We can cluster the companies which are part OTH to the following groups: network operators (like T-Mobile, China Mobile and Sprint Nextel), software developers (like eBay, NXP, Google and PacketVideo), device manufacturers (like HTC, LG, Samsung, ZTE, Dell and Lenovo), component manufacturers (like ARM, Freescale Semiconductor, Texas Instruments and Nvidia) and others (like Flex Comix and Borqs). By the way, OHA members are not allowed to produce devices which are based on forks of the Android OS[10] - logos of some of the member companies are shown below[11].

Lastly, the companies which are part of OTH have allocated significant engineering resources to improve Android and bring Android devices to market. The companies that have invested in Android have done so because they believe an open platform is necessary. Thus, a group of organizations with shared needs has pooled resources to collaborate on a single implementation of a shared product. The objective is a shared product that each contributor can tailor and customize[12].

[9] https://www.openhandsetalliance.com/
[10] https://en.wikipedia.org/wiki/Open_Handset_Alliance
[11] https://slideplayer.com/slide/11753561/
[12] https://source.android.com/docs/setup/about

Android Auto

“Android Auto” allows us to use our own applications on the car display while we connect our mobile phone to the infotainment system - as shown below[13]. “Android Auto” is supported on over 500 car’s models such as: Audi (2016+), Aston Martin (2018+), Mercedes-Benz (2017+), Mitsubishi (2016+), Lamborghini (2016+), Lexus (2020+), RAM (2018+), Honda (2016+) and more[14].

Overall, “Android Auto” lets us use our voice to perform operations without the need of installing\downloading any third party software. For it to work we need an Android 8.0+, data plan and car/stereo compatible with Android auto. For connecting an Android device we can use a USB cable or even a wireless connection if the device and car supports that[15].

Lastly, it is important to understand that “Android Auto” and “Android Automotive” are two different things[16]. Thus, “Android Auto” is a smart driving companion helping us stay focused and entertained with the Google Assistant. It is designed for making it easier to use apps while we are on the road[17].

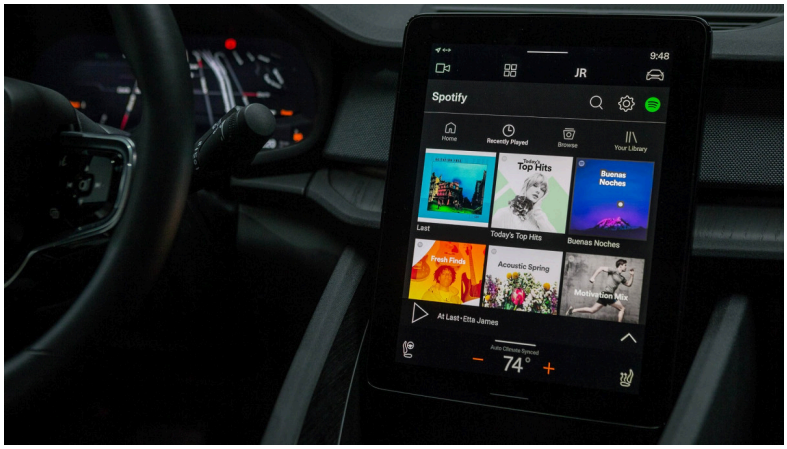
[13] https://www.trendradars.com/channels/article-923412-how-to-take-screenshots-on-android-auto/
[14] https://www.android.com/auto/compatibility/vehicles/
[15] https://www.android.com/auto/#auto-feature-carousel
[16] https://source.android.com/docs/automotive/start/what_automotive
[17] https://play.google.com/store/apps/details?id=com.google.android.projection.gearhead&hl=en

Android Automotive (AAOS)

“Android Automotive” (AAOS) is an open-source operating system based on Android which is designed for vehicle dashboards (in-vehicle infotainment system aka IVI systems) - as shown below[18]. AAOS was developed by Google and Intel in conjunction with car manufacturers like Audi and Volvo (introduced in 2017). The goal is to create an operating system codebase for vehicle manufacturers to develop their own version\distribution for handling infotainment taks (messaging, navigation, music playback, controlling air-condition and more)[19].

Overall, as opposed to “Android Auto”[20] AAOS is a full operating system executing on the car itself without the need for a smartphone in order to work. It also supports applications built for Android as well those built for “Android Auto”. Automotive OEMs can also license and integrate GAS (Google Automotive Services) which is similar to GMS (Google Mobile Services) on smartphones. GAS provides applications and services like “Google Maps” and “Google Assistant”[21].

Lastly, “Android Automotive” and “Android Auto” are two distinct products. AAOS is an operating system while “Android Auto” is an application. Thus, “Apple CarPlay” is a direct competitor of “Android Auto” and not AAOS. “Android Automotive” is included in the AOSP[22]. Hence, using AAOS provides different benefits for car manufactures like (not limited to): it’s familiar, flexible and backed by Google[23].



[18] https://www.androidworld.it/aggiornamenti/android-automotive-os-12l.html
[19] https://en.wikipedia.org/wiki/Android_Automotive
[20] https://medium.com/@boutnaru/the-android-concept-journey-android-auto-4d71fd9d03f8
[21] https://source.android.com/docs/automotive/start/what_automotive
[22] https://medium.com/@boutnaru/the-android-concept-journey-aosp-android-open-source-project-deca886302ff
[23] https://www.withintent.com/blog/choosing-android-automotive/

API Level (Application Programming Interface Level)

An API Level (Application Programming Interface Level) is basically an integer number. This number uniquely identifies the framework API (packages/classes/etc) revision offered by a specific version of the Android platform. We use the API level for describing the minimum (minSdk)\maximum API revision that an application requires\allows. Hence, Android can install only supported applications and block others[24].

Overall, it is important to understand that specific packages\classes\functions can be available only from a specific API level. For example NdkBinder and Native MIDI API (AMidi) are available only from API Level 29 (Android 10) or later[25]. Each API level is relevant for a specific version of Android - as shown in the parietal table below[26].

Lastly, in the case of Android (which is not true for other environments) the SDK version is practically a synonym to API level - also shown in the table below. Thus, every Android application specifies a targetSdkVersion (aka the target API level) in the manifest file. This is done to provide a context of how the developer expects the application to execute[27].

Version	SDK / API level	Version code	Codename	Cumulative usage [1]	Year [4]
Android 16 BETA	Level 36	BAKLAVA	Baklava [2]	0%	*TBD*
Android 15	Level 35	VANILLA_ICE_CREAM	Vanilla Ice Cream [2]	1.34%	2024
Android 14	Level 34	UPSIDE_DOWN_CAKE	Upside Down Cake [2]	38.32%	2023
	▪ `targetSdk` must be 34+ for new apps and app updates as of August 31, 2024.				
Android 13	Level 33	TIRAMISU	Tiramisu [2]	57.1%	2022
Android 12	Level 32 Android 12L	S_V2	Snow Cone [2]	70.3%	
	Level 31 Android 12	S			2021
Android 11	Level 30	R	Red Velvet Cake [2]	82.2%	2020
Android 10	Level 29	Q	Quince Tart [2]	88.5%	2019
Android 9	Level 28	P	Pie	92.2%	2018
Android 8	Level 27 Android 8.1	O_MR1	Oreo	93.4%	2017
	Level 26 Android 8.0	O		95.7%	
Android 7	Level 25 Android 7.1	N_MR1	Nougat	96.0%	2016
	Level 24 Android 7.0	N		97.0%	
Android 6	Level 23	M	Marshmallow	98.4%	2015
Android 5	Level 22 Android 5.1	LOLLIPOP_MR1	Lollipop	98.8%	
	Level 21 Android 5.0	LOLLIPOP, L		99.7%	2014
	▪ Jetpack/AndroidX libraries require a `minSdk` of 21 or higher since April 2024. ▪ Jetpack Compose requires a `minSdk` of 21 or higher. ▪ Google Play services v23.30.99+ (August 2023) drops support for API levels below 21.				
Android 4	Level 20 Android 4.4W [3]	KITKAT_WATCH	KitKat	99.9%	
	Level 19 Android 4.4	KITKAT			2013
	▪ Jetpack/AndroidX libraries require a `minSdk` of 19 or higher since October 2023. ▪ Google Play services v21.33.56+ (July 2021) drops support for API levels below 19.				
	Level 18 Android 4.3	JELLY_BEAN_MR2	Jelly Bean	99.9%	
	Level 17 Android 4.2	JELLY_BEAN_MR1		99.9%	2012
	Level 16 Android 4.1	JELLY_BEAN		99.9%	
	▪ Google Play services v14.8.39+ (December 2018) drops support for API levels below 16.				

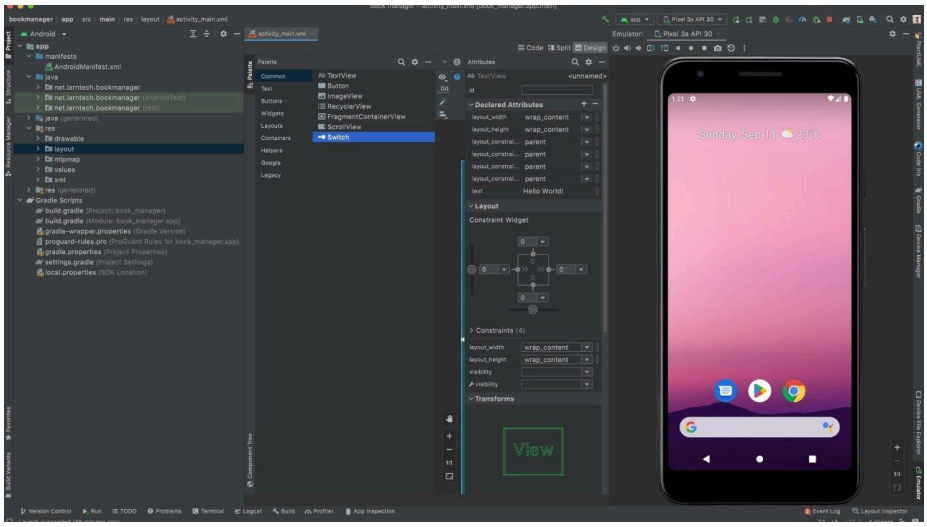
[24] https://www.dre.vanderbilt.edu/~schmidt/android/android-4.0/out/target/common/docs/doc-comment-check/guide/appendix/api-levels.html
[25] https://atsushieno.github.io/2022/11/12/ndk-conditional-code-per-api-level.html
[26] https://apilevels.com/
[27] https://support.google.com/googleplay/android-developer/answer/11926878?hl=en

Android Emulator

The goal of an Android emulator is to simulate Android devices (mobile phone/tablet/watch/etc) on a personal computer. By doing so we can test our application on various devices\API levels[28] without the need for dedicated physical devices - an example is shown in the screenshot below[29].

Overall, we can summarize the flow of working with an emulator to the following phases. First, verifying system requirements (for best experience at least 16 GB RAM, 16 GB disk space and 64-bit version of Windows 10+\MacOS 12+\Linux\ChromeOS). Second, creating an AVD (Android Virtual Device) configuration - more on that in a future writeup. Third, run our app on the emulator by pushing it manually or from the IDE (like “Android Studio”) - also shown below. Fourth, navigate and use the emulator[30].

Lastly, it is import to know that there are multiple emulators for Android such as: “BlueStacks”, “Android Studio”, “Bliss OS”, “GameLoop”, “PrimeOS”, “NoxPlayer”, “MeMUPlay” and “LDPlayer 9”[31]. Beside emulators we also have Android VM (Virtual Machines) but more on those in future writeups.



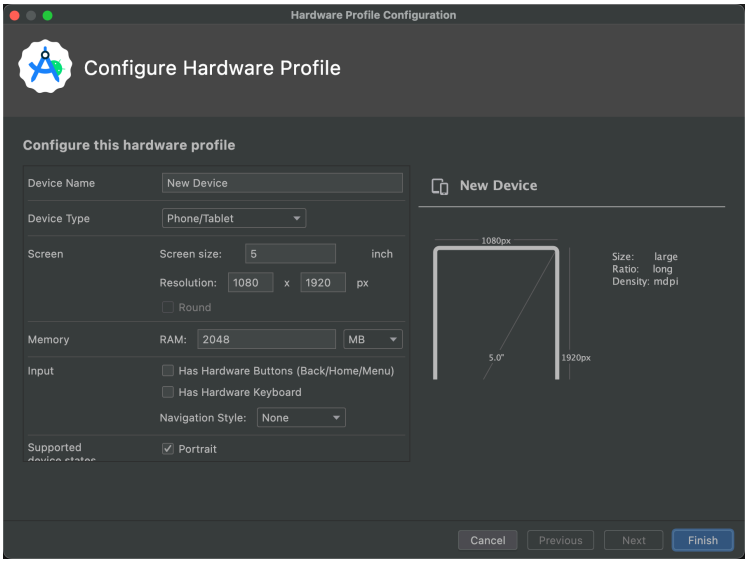
[28] https://medium.com/@boutnaru/the-android-concept-journey-api-level-application-programming-interface-level-df4cee3f85c9
[29] https://serwer2311392.home.pl/Simulador-Android/emulator-build-and-run-your-project-on-android-emulator-1629738/
[30] https://developer.android.com/studio/run/emulator
[31] https://www.androidauthority.com/best-android-emulators-for-pc-655308/

AVD (Android Virtual Device)

The goal of AVD (Android Virtual Device) is to provide configuration which defines the characteristics of an Android based device (phone\tablet\Wear OS\Android TV\Automotive OS). This is done in order to simulate the device using the "Android Emulator"[32].

Overall, We can create an AVD using the "Device Manager" launched from "Android Studio" (View -> Tool Windows -> Device Manager) - it can also be launched\used from other IDEs. It is important to know that an AVD consists of: hardware profile, system image, storage area (user data, installed apps and emulated SD card), skin (appearance of the device) and more. By the way, beside the predefined hardware profiles for common devices which are part of the device manager we can also create our own - as shown in the screenshot below[33].

Lastly, we can create an AVD for a specific Android version. Also, we can simulate different network speeds/latency as part of our AVD. Due to the fact everything is virtual we can of course create multiple AVDs and even run them if we have sufficient resources[34].



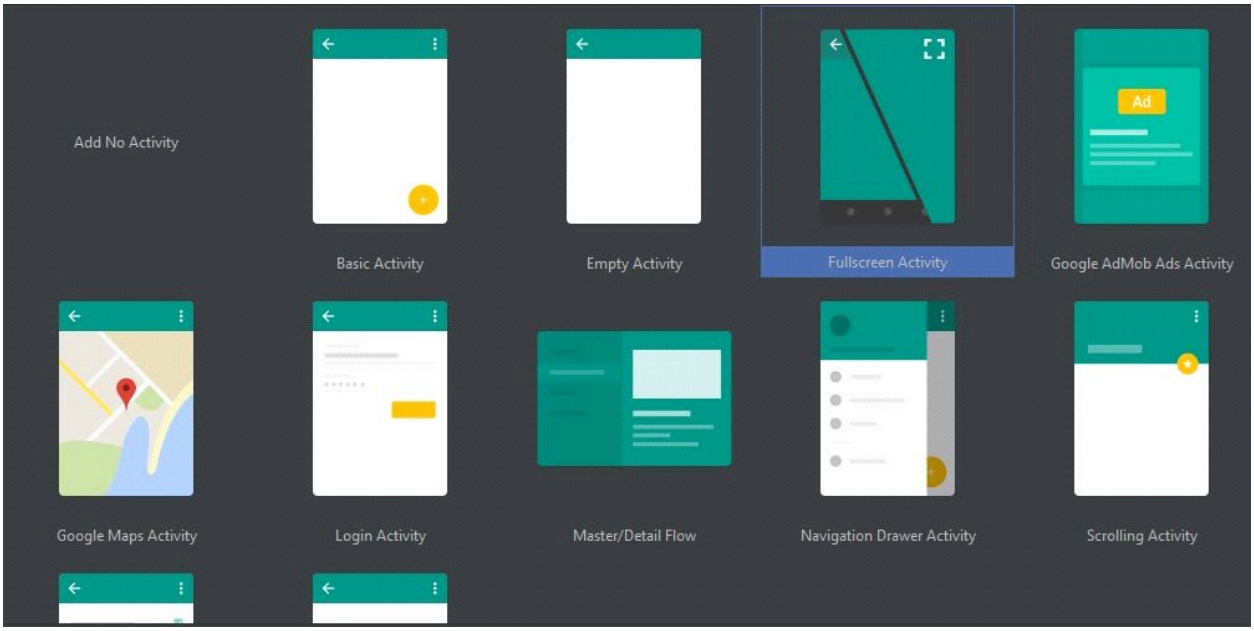
[32] https://medium.com/@boutnaru/the-android-concept-journey-android-emulator-2e53cc3a6d26
[33] https://developer.android.com/studio/run/managing-avds
[34] https://www.sitepoint.com/beginning-andoid-create-an-android-virtual-device/

Activity

The Android application model is based on activities, which are launched and put together as one unit. As opposed to other executable formats (like ELF and PE) an APK (Android Package) file does not have a "main()". Thus, the Android system calls code in a specific Activity (which is defined in the manifest file as the "entry point" of the application). Also, an activity provides the windows where the application draws its user interface[35] - as shown in the screenshot below taken from the "Android Studio" IDE[36].

Overall, during the lifetime of the application activities change their state. In case of a change state one of the following callback is triggered: "onCreate()", "onStart()", "onRestart()", "onResume()", "onPause()", "onStop()" and "onDestroy()"[37] - more information on each in future writeups.

Lastly, using activities help in creating user flows and saving the context in case the process is killed. Thus, when the user returns to the activities their previous state can be restored[38]. An activity as a subclass of the "Activity" class[39].



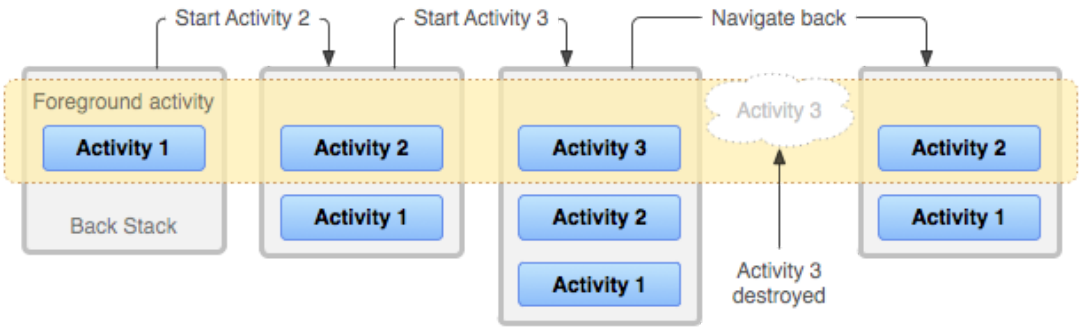
[35] https://developer.android.com/guide/components/activities/intro-activities
[36] https://www.includehelp.com/android/use-full-screen-activity-in-android-studio.aspx
[37] https://www.geeksforgeeks.org/activity-state-changes-in-android-with-example/
[38] https://developer.android.com/guide/components/fundamentals
[39] https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/app/Activity.java

Back Stack

In case we start a new application\move to a new activity[40] the previously visible activity is moved to the "activity back stack". Thus, we can say that the "activity back stack" holds a collection of activities that have been suspended and can easily be resumed. Every process\application has its own back stack. An activity can be removed from the back stack if we close it, the user/application/OS calls the finish() function[41] - as shown in the screenshot below.

Overall, activities in the back stack are never rearranged. Hence, if our application allows the creation of an activity from one of more activities, a new instance of the activity is created and pushed to the stack[42].

Lastly, by default when the user clicks the "back button" every instance of the activity is shown in the order in which they were opened (unless they were closed\destroyed). Also, the activities are shown with their UI state[43].



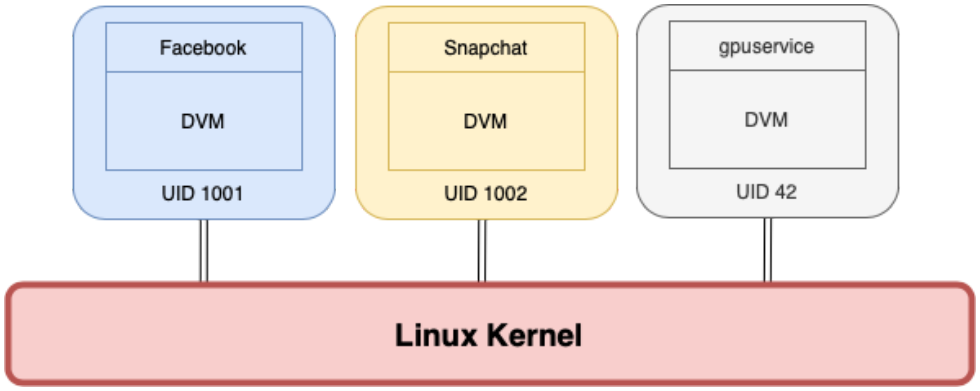
[40] https://medium.com/@boutnaru/the-android-concept-journey-activity-74314ee42ce1
[41] https://joshuadonlan.gitbooks.io/onramp-android/content/intents/activity_stack.html
[42] https://developer.android.com/guide/components/activities/tasks-and-back-stack
[43] https://guides.codepath.com/android/Navigation-and-Task-Stacks

Application Sandbox

Android isolates applications from each other. This is done by leveraging operating based user management. Android assigns a unique UID[44] for every Android application upon installation. By doing so Android set up a kernel-level "Application Sandbox"[45].

Moreover, every android application gets its own process (in conjunction with a different UID), which provides a different memory address space[46] - as shown in the diagram below. The sandbox design has been part of Android since its creation. It leverages Linux kernel features such as (but not limited to): memory isolation (DAC), file isolation, UIDs, GIDs and in newer versions also namespaces[47].

Lastly, when creating an android package (APK) we can define as part of the manifest we want to share a UID with another similarly-signed APKs[48].



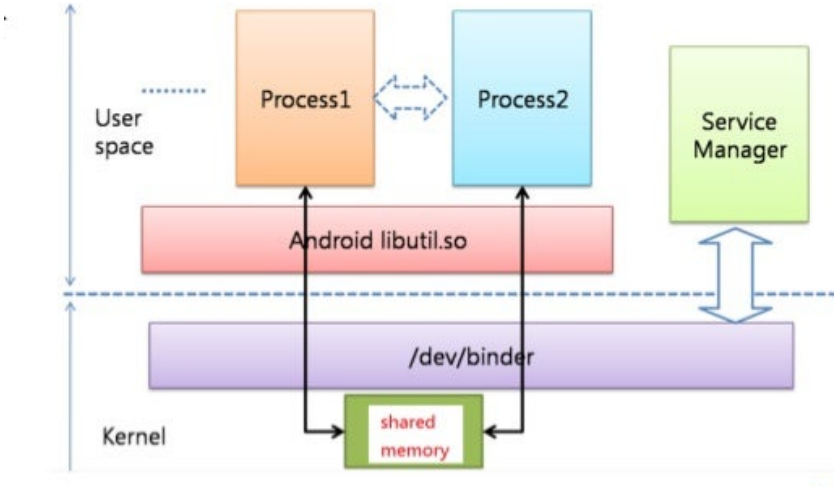
[44] https://medium.com/@boutnaru/the-linux-security-journey-uid-user-identifier-2f11bcf90ee8
[45] https://source.android.com/docs/security/features
[46] https://stackoverflow.com/questions/31531633/what-is-a-application-sandwork-in-android-and-how-it-works
[47] https://2net.co.uk/slides/sandbox-csimmonds-droidcon-london-2023.pdf
[48] https://source.android.com/docs/security/overview/app-security

Binder

Due to the application sandbox (which is a fundamental part of the Android architecture) every application is its own process with a separate memory address space[49]. Hence, for passing information between applications we have to leverage some kind of an IPC (Inter Process Communication) mechanism[50].

Overall, Android does not use any SysV IPC functionally which is part of Linux[51]. For that we have Binder, which is an Android-specific interprocess communication mechanism, and remote method invocation system - as shown in the diagram below[52]. Using the binder component one Android process can call a routine in another Android process. Binder helps in identifying the method to invoke and passing the arguments between processes[53].

Lastly, the Binder framework used as part of the Android operating system is based on the OpenBinder project. The kernel side of OpenBinder was merged into the Linux kernel mainline as part of version 3.19[54]. Thus, we can also checkout this source code as part of the Linux kernel[55].



[49] https://medium.com/@boutnaru/the-android-concept-journey-application-sandbox-c5cb80480fd3
[50] https://medium.com/@boutnaru/ipc-manager-inter-process-communication-manager-348930d08fa0
[51] https://medium.com/@boutnaru/linux-namespaces-ipc-namespace-927f01cbcf3d
[52] http://www.slideshare.net/magoroku15/android-binderipc
[53] https://elinux.org/Android_Binder
[54] https://en.wikipedia.org/wiki/OpenBinder
[55] https://elixir.bootlin.com/linux/v6.14.1/source/drivers/android/binder.c