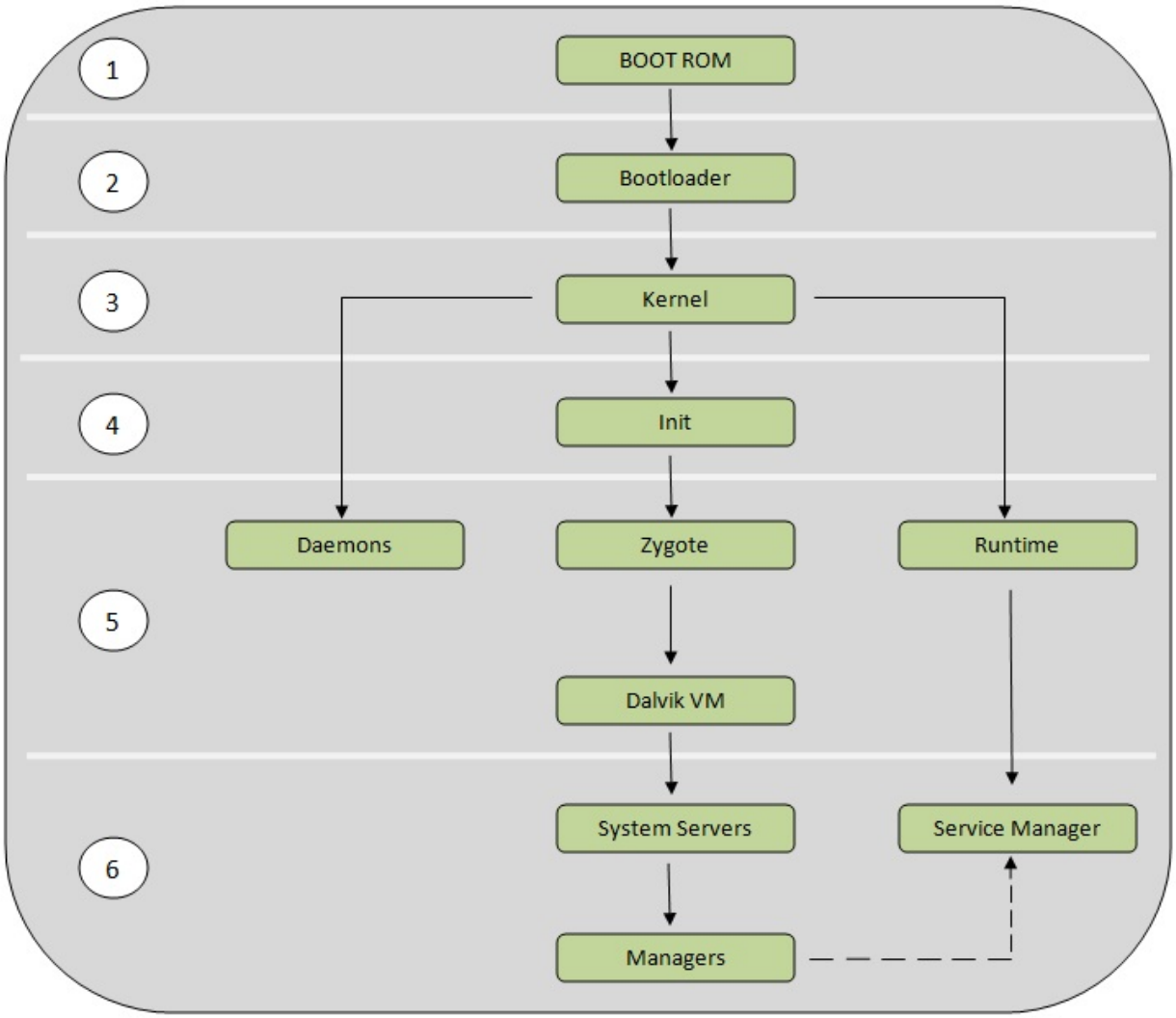
Android boot process:

In computing, booting is starting up a computer or computer appliance until it can be used. It can be initiated by hardware such as a button press, or by software command. After the power is switched on the computer is relatively dumb, and can read only part of its storage called Read-only memory. There a small program is stored called firmware. It does power-on self-tests, and most importantly, allows accessing other types of memory, like hard disk and main memory. The firmware loads bigger programs into the computer's main memory and runs it. In general purpose computers, but as well in smart phones, tablets, optionally a boot manager is run.

Consider the following graph:



Step 1: Power On and System Startup

When we press the power button, the Boot ROM code starts executing from a pre-defined location which is hardwired in ROM. It loads the Bootloader into RAM and starts executing.

Step 2: Bootloader

The bootloader is a small program which runs before Android does. This is NOT part of the Android operating system. The bootloader is the place where manufacturer puts their locks and restrictions.

The bootloader executes in two stages. In the first stage it detects external RAM and loads a program which helps in the second stage.

In the second stage, the bootloader setups the network, memory, etc, which requires to run kernel. The bootloader is able to provide configuration parameters or inputs to the kernel for specific purposes.

The bootloader can be found at:

<android source>/bootable/bootloader/legacy/usbloader

This legacy loader contains 2 important files:

1. 1- Init.s :: Initializes stacks, zeros the BSS segments and call_main() in main.c
2. 2- Main.c :: Initializes hardware (clocks, board, keyboard, console) and creates linux tags.

Step 3: Kernel

The Android kernel starts in a similar way as the linux kernel. As the kernel launches, is starts to setup cache, protected memory, scheduling and loads drivers. When the kernel finishes the system setup, it looks for “init” in the system files.

What is the difference between the linux and android kernels?, here's a list of changes/addons that the Android Project made to the Linux kernel:

- Binder: It is an Android specific interprocess communication mechanism and remote method invocation system.
- ashmem: "Android Shared Memory". It is a new shared memory allocator, similar to POSIX SHM but with a different behavior and sporting a simpler file-based API.
- pmem: "Process memory allocator": It is used to manage large (1-16+ MB) physically contigous regions of memory shared between userspace and kernel drivers.
- logger: This is the kernel support for the logcat command.
- wakelocks: It is used for power management files. It holds the machine awake on a per-event basis until wakelock is released.
- oom handling: It kills processes as available memory becomes low.
- alarm manager: It lets user space tell the kernel when it would like to wake up.
- RAM_CONSOLE: Allows to save kernel printk messages to a buffer in RAM, so that after a kernel panic they can be viewed in the next kernel invocation.
- USB gadget driver for ADB
- yaffs2 flash filesystem

Step 4: init process

Init is the very first process, we can say it is a root process, or the grandfather of all processes. The init process has two responsibilities.

1- Mounts directories like /sys , /dev or /proc

2- Runs init.rc script

- - The init process can be found at /init :: <android source>/system/core/init
- - Init.rc file can be found at :: <android source>/system/core/rootdir/

Android has specific format and rules for init.rc files. More information about this rules can be found in: What is inside the init.rc and what is it used for.

At this stage, you can finally see the Android logo in your screen.

Step 5: Zygote and Dalvik

In Java, we know that a separate Virtual Machine instance will popup in memory for separate per app, but in the case of Android, the VM should run as quick as possible for an app. But what happens if you have several apps thus launching several instances of the Dalvik (VM)?, it would consume an immense amount of memory.

To overcome this problem, the Android OS has a system called “Zygote”. The Zygote enables code sharing across the Dalvik VM, achieving a lower memory footprint and minimal startup time. Zygote is a virtual machine process that starts at system boot. The Zygote preloads and initializes core library classes.

The Zygote loading process:

1. Load Zygote Init class:
   <android source>/frameworks/base/core/java/com/android/internal/os/ZygoteInit.java
2. registerZygoteSocket() :: It registers a server socket for zygote command connections.
3. preloadClasses() :: Is a simple text file that contains a list of classes that need to be preloaded, you can find the file at <android source>/framework/base
4. preloadResources() :: Everything that is included in the android.R file will be loaded with this method (themes and layouts).

At this time, you can see the boot animation.

Step 6: System service

After the above steps are completed, Zygote launches the system services. The Zygote forks a new process to launch the system services.

Core services:

- Starting power manager
- Creating the Activity Manager
- Starting telephony registry
- Starting package manager
- Set activity manager service as system process
- Starting context manager
- Starting system contact providers
- Starting battery service
- Starting alarm manager

- Starting sensor service
- Starting window manager
- Starting Bluetooth service
- Starting mount service

Other services:

- Starting status bar service
- Starting hardware service
- Starting NetStat service
- Starting connectivity service
- Starting Notification Manager
- Starting DeviceStorageMonitor service
- Starting Location Manager
- Starting Search Service
- Starting Clipboard Service
- Starting checkin service
- Starting Wallpaper service
- Starting Audio Service
- Starting HeadsetObserver
- Starting AdbSettingsObserver