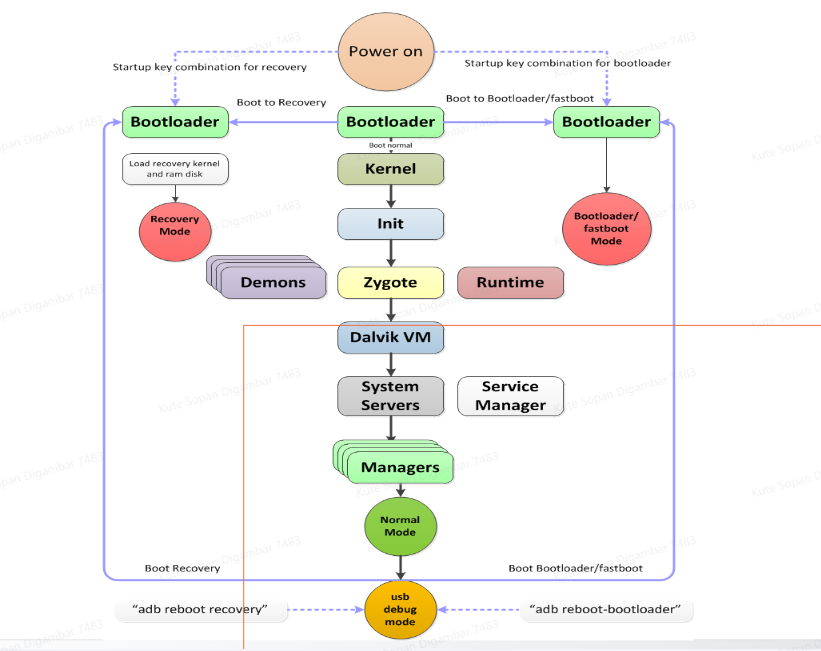


****

**Android Booting Sequence**

* When switching off the power of the Android device and switch on it again, this process is known as the Android Booting sequence.

**Step 1: Boot ROM**

* Whenever we press the power button, the "BIOS" code starts executing from a pre-defined location which is exists in the ROM memory on motherboard.
* BIOS ( Basic Input Output System ) which is a micro-program. A micro-program is a program which is smaller in size.
* first step of BIOS is "POST" - Power On Self Test , under POST BIOS checks whether all peripherals are connected properly or not and their working status.
* It loads the Bootloader into RAM and starts executing.

**Step 2: Bootloader**

* The Bootloader is a first small program which runs before Android System and responsible for bringing up the kernel on a device. Bootloader is hardware specific and it is NOT part of the Android specific.
* Bootloader is low-level code contains instructions that tell a device how to start up and find system Kernel. The bootloader is the place where manufacturer puts their locks and restrictions. It is highly secured coz it figures out where the kernel image is to be loaded from.
* The Bootloader executes in two stages.
* First Stage

it detects external RAM and loads a program. The kernel decompresses itself and starts executing into memory. in First-stage Bootloader will provide support for loading Recovery image to the System flash. Load the boot images, such as boot.img, vendor\_boot.img, init\_boot.img etc.

* Second Stage

Bootloader loads Kernel from Boot-partition. the bootloader setups the System files, network, memory, etc, which requires to run kernel. The bootloader is able to provide configuration parameters or inputs to kernel for specific purposes.

* Android devices include several partitions that serve different functions in the boot process. There are different Boot partitions: Cache, Misc., UserData, Boot, System

The bootloader can be found at:

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

This legacy loader contains 2 important files:

1- Init.s :: Initializes stacks, zeros the BSS segments and call\_main() in main.c

2- Main.c :: Initializes hardware (clocks, board, keyboard, console) and creates Linux tags.

* BSS refers to uninitialized global and static objects. mapped into memory that is initialized to zero.

**Step 3: Kernel**

**What is Kernel ?**

* The kernel is a computer program at the core of a computer's operating system and generally has complete control over everything in the system.
* It is the portion of the operating system code that is always present in memory and facilitates interactions between hardware and software components.
* A kernel controls all hardware resources (e.g. I/O, memory ) via device drivers.
* On most systems, the kernel is one of the first programs loaded on start-up (after the bootloader). It handles the rest of start-up as well as memory, peripherals, and input/output (I/O) requests from software.
* The Android kernel is based on an upstream Linux Long Term Supported (LTS) kernel. At Google, LTS kernels are combined with Android-specific patches to form what are known as Android Common Kernels (ACKs).

**Android Kernel :**

* The Android kernel starts in a similar way as the Linux kernel. Once kernel Boots, it starts to setup cache, protected memory, scheduling, loads drivers, starts kernel daemons. 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 behaviour and sporting a simpler file-based API.

(The POSIX shared memory API allows processes to communicate information by sharing a region of memory.)

* + Pmem : "Process memory allocator" It is used to manage large (1-16+ MB) physically contiguous regions of memory shared between userspace and kernel drivers.
  + Logger : This is the kernel support for the logcat command.
  + wake locks : It is used to make efficient use of power management. It

holds the machine awake on a per-event basis until wake

lock is released.

* + oom handling : It kills processes as available memory becomes low.

(OutOfMemory Error)

* + 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, it is a root process of all processes. Main function is To get the core UserSapce functionalities up and Running.
* It is a program to initialize the elements of the Android system. Unlike Linux, Android uses its own initialization program.
* The init process has two responsibilities.

1. Creates many directories like /sys, /dev or /proc

2. Runs init.rc script. - provides the generic initialization instructions

- Starts Native Daemons : usdb, rild, adbd etc.

- Start Zygote Process .

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

init <machine name>.rc - provides specific initialization instructions also it is

imported by the init.rc program. (machine name is the name of the hardware that Android is running on)

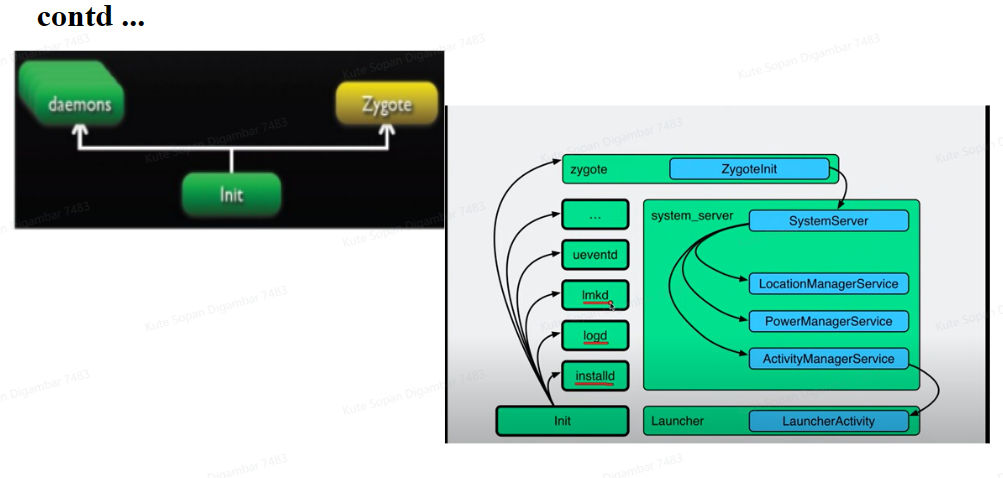
* The init process can be found at (Vender Specific) :

<android source>/system/core/init

* Init.rc file can be found at (System Specific) :

<android source>/system/core/rootdir/

PID for init process = 1.



**Step 5: Zygote and Dalvik**

* Zygote is a VM process in Android that starts at system boot. which handles the forking of each new application process. Each new Zygote process is a child of the original Zygote process and includes a VM.
* When the command “app\_process” launches the Zygote , the first VM instance is created. The VM then calls Zygote’s main () function.
* Zygote forks itself to form a base for other Applications and the application is then bound to the thread of this process. Thus each application runs in its very own Linux process, with its own thread.
* Zygote to preload all shared Java classes , resources and libraries into memory. Also forking itself to create SystemServer. Zygote and Dalvik VM is written in Java.
* the Android OS has a system called “Zygote”. The Zygote enables code sharing across the Dalvik VM, achieving a lower memory footprint and minimal start-up time.

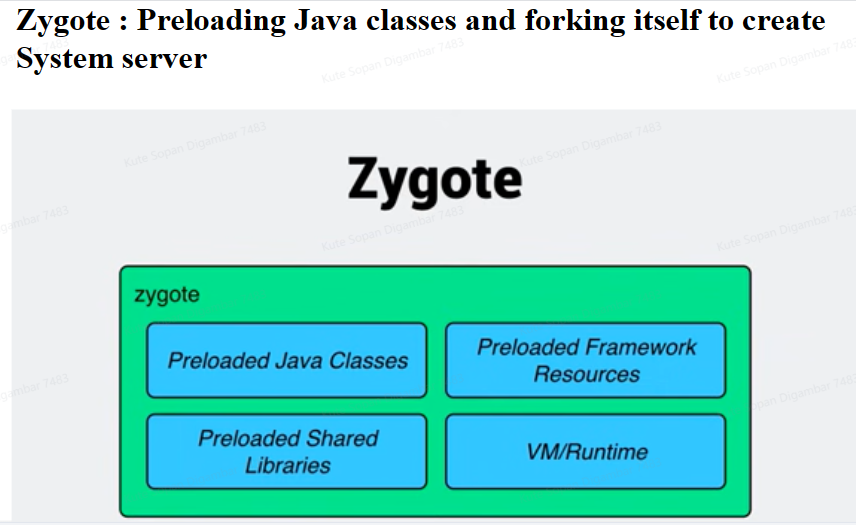
The Zygote loading processes:

Load Zygote Init class:

<android source>/frameworks/base/core/java/com/android/internal/os/ZygoteInit.java

* registerZygoteSocket() :: It registers a server socket for zygote command connections.
* 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/bas
* 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 Zygote it launches the System Server. The Zygote forks a new process to launch the system services.
* The System server is the core the Android system and it is started as soon as Dalvik is initialized and running. At first server will load a native library android\_servers that provides interfaces to native functionalities.
* SystemServer contains many services such as Activity manager, Package manager, Window manager etc. that are mostly written in Cpp also contains Surface flinger, audio flinger etc that are written in mostly C/Cpp .
* Then the native init method that will set up native services is called.
* After setting-up the native services it creates the server thread. this thread will start the remaining services in system as according to necessary.
* Each service is running in a separate Dalvik thread in the SystemServer process . Once system services up and running in Memory, Android has Completed boot process. After "ACTION\_BOOT\_COMPLETED" standard broadcast action will fire.

Now we have finally completed the booting process (system service are up and running in memory).

Need to analyse the Android Bootup?

The logcat :: Use adb to get the booting process events from the logcat.

‘adb logcat –d –b events | grep “boot”

‘adb logcat –d | grep preload’

