International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 4 Issue: 4 616 - 618

Android Boot Time Optimization Techniques

Mr. Chirag Soni
Student, Bachelor of Engineer
K. J. Somaiya College of Engineering
Mumbai, India

Mr. Sahil Shah
Student, Bachelor of Engineer
K. J. Somaiya College of Engineering
Mumbai, India

Mr. Shrinath Apake
Student, Bachelor of Engineer
K. J. Somaiya College of Engineering
Mumbai, India

Prof. Dipti Pawade
Assistant Professor, Department of IT
K. J. Somaiya College of Engineering Mumbai, India

Abstract **-** Additional Software System Initialization and configuration throughout startup is required to cope up with increasing application needs in embedded systems that adversely affects system boot time. Quick boot is crucial for consumer devices like mobile phones, tablets, digital TV, etc. This paper, in the beginning, gives you an idea about booting, boot time, measuring boot up time & booting process of an android device. Then various approaches like Hibernation Based Boot & Boot Optimization are discussed as well as compared to minimize the boot time. Then to cut back startup time ,we discuss techniques such as Optimizing X-loader, U-Boot, Boot-0 & Boot-1, Linux Kernel, Android Framework which reduces the total startup time to great extent.

Index Terms **-** *Boot time, Android Boot process, Hibernation Based Boot, Boot time optimization.*

*****

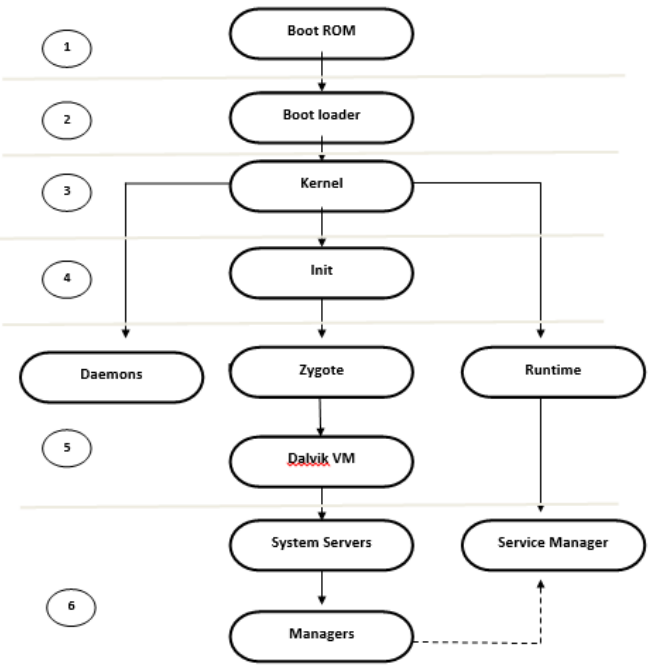
I. INTRODUCTION

In registering, booting is the introduction of a modernized framework. The framework can be a PC or a PC machine. The booting procedure can be "hard", after electrical energy to the CPU is changed from off to on (keeping in mind the end goal to analyze specific equipment mistakes), or "delicate", when those power-on self-tests (POST) can be evaded. Delicate booting can be started by equipment, for example, a catch press, or by programming order. Booting is finished when the ordinary, agent, runtime environment is accomplished.
The time that an item takes to boot is known as boot time. It straightforwardly affects the primary impression of end client for the item. Despite how appealing or all around planned a customer electronic gadget is, the time required to move the gadget from off to an intuitive, usable state is basic to getting a positive end client experience. Boot time can be measure by considering following parameters [3]

- Printk Times - Simple system for showing timing information for each printk.
- Kernel Function Trace - System for reporting function timings in the kernel.
- Linux Trace Toolkit - System for reporting timing data for certain kernel and process events.
- Oprofile - System-wide profiler for Linux.
- Bootchart - A tool for performance analysis and visualization of the Linux boot process. Resource utilization and process information are collected during the user-space portion of the boot process and are later rendered in a PNG, SVG or EPS encoded chart.
- Bootprobe - A set of System Tap scripts for analyzing system bootup.
- Grabserial - a nice utility from Tim Bird to log and timestamp console output
- Process trace - a simple patch from Tim Bird to log exec, fork and exit system calls.

II. BOOTING PROCESS

Figure 1 explains the flow of booting process. Steps involved in booting process of an android device are [2][8]:



Step 1: Power On & System Startup

When we press the ability button, the Boot computer storage code starts execution from a pre-defined location that is hardwired in computer storage. It loads the Boot loader into RAM and starts execution.

Step 2: *Boot loader*

The boot loader is a tiny program that runs before android starts. This is NOT a part of the android operating system. The boot loader is that place where manufacturer puts their locks and restrictions. The boot loader executes in 2 stages. Within the initial stage it detects external RAM and loads a program that helps within the second stage. To run kernel, the boot loader setups the memory, network, etc. in the second stage. The boot loader is ready to produce configuration parameters or inputs to the kernel for specific functions.

The boot loader is found at:

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

This bootloader contains two necessary files:

1. Init.s: Initializes stacks, zeros the BSS segments2 and calls main () method in main.c
2. Main.c: It creates Linux tags & Initializes hardware (keyboard, console, clocks, board)

Step 3: Kernel

The android kernel starts in a very similar manner as the UNIX kernel. As the kernel launches, it starts to setup cache, protected memory, planning and loads drivers. After that init process is searched which plays vital role in next step.

Step 4: init process

init is that the very 1st method, we will say it's a root method of all the processes. The init method has 2 responsibilities.

- Mounts directories like /sys, /dev, /proc
- Runs init.rc script

- You can find init method at /init:

<android source>/system/core/init

-You can find Init.rc file at:

<android source>/system/core/rootdir/

Android has specific format and rules for init.rc files.

Step 5: Zygote and Dalvik

In Java, for each application a separate Virtual Machine instance can popup in memory , however in the case of android, the VM ought to run as fast as attainable for an application. But, what happens if you have got many apps that launches many instances of the Dalvik (VM)? It'd consume a large quantity of memory.

Solution to this problem is provided by Zygote system. The zygote allows code sharing across the Dalvik VM, achieving a lower memory footprint and nominal startup time. Zygote is a virtual machine method that starts at system boot. The zygote preloads and initializes core library classes. Following are the key methods of Zygote

- registerZygoteSocket() : A server socket is registered by registerZygoteSocket() for zygote command connections.
- preloadClasses() : It is a file that contains a listing of classes that require to be preloaded, it is found at <android source>/framework/base
- preloadResources() : Everything that's enclosed within the android.R file is loaded with this technique (themes and layouts).

Step 6: *System Service*

After the above steps are completed, system services are launched by zygote . The zygote forks a brand new process to launch the system services. Some of the core services like Activating power manager, creating Activity manager, starting telephone register, Starting package manager,etc. & also other services like Activating status bar service, Activating hardware service, Activating NetStat service, Starting connectivity service,etc.

Step 7: Boot Completed

Once System Services up and running in memory, Android has completed booting process, At this time "ACTION_BOOT_COMPLETED" standard broadcast action will fire.

III. APPROACHES TO MINIMIZE BOOT TIME

Hibernation based boot and boot sequence optimization are the two important approaches to minimize the booting time of android. Difference between Hibernation based boot and boot sequence optimization is given in table 1

TABLE I

HIBERNATION BASED BOOT AND BOOT SEQUENCE OPTIMIZATION

Hibernation Based Boot [4]	Boot Optimization [7]
Snapshot of memory image of a fully booted Android device is taken.	Optimize every component responsible for boot up.
When users start their devices, the device will start from hibernation mode instead of normal boot up.	This is widely used technique over Hibernation Based Boot.
The boot up sequence is modified, where the snapshot image is loaded and then the device is stared, rather than the core startup.	In this stage, we modify two files init.rc and Zygote class preloading.
The main disadvantage of this approach is to maintain the latest snapshot image. Thus, whenever an application is installed on the tablet and user wants to start that application during boot time; the snapshot needs to be recreated.	The disadvantage of this approach is the increase in memory usage while avoiding Zygote preloading. The next stage will overcome this disadvantage and will also reduce boot up time without affecting the memory usage.

As discuss in section II android boot process involves activities like X-loader, U boot, kernel, Zygote and Dalvik. Even if time requirement for each step in boot process is reduced to some extents, the total boot up time can be optimized to great extent. Following are the ways to optimize android boot time

Optimizing X-loader

Default setup contains numerous segments which may not be helpful from end client perspective, such components like

NAND bolster, one NAND bolster and so on can be expel from the arrangement. As there are few backings, the aggregate time required for design can be minimized to some level. Another path is to evacuate the investigating base and traceability code when item is handover to end client. Processor clock rate assumes a basic part in enhancing X-loader. The booting time can be let down if processor clock is set to the most extreme conceivable rate. [8]

Optimizing U-boot

Like x-loader optimization, U boot time can be lower down by eliminating modules of the default configuration. This may include Ethernet support, USB support etc. Use of simple parser in place of hush can also lower down the boot time. Verbose message printed during initialisation also contribute to some extents to overall boot time. Providing this facility to end user doesn't make sense, so it can be disabled. For reducing U boot timing, first we need to raise the throughput by optimizing NAND [8].

Optimising Boot-0 & Boot-1

Boot 0 and Boot 1 includes activities like loading DRAM, initializing LCD panel, NAND flash, HDMI module etc.The total time required to complete this stage is approximately 1.7 sec. This time can be lower down to 0.9 sec by eliminating initialization of these peripheral modules [2].

Optimising Linux Kernel

To reduce Kernel bootup time unused modules in kernel should be removed and a number of drivers as modules needs to be build up and loaded. One can also remove debug info from the kernel modules. Removing unwanted delays in a number of the kernel modules additionally helped cut back the kernel boot up time to certain extent [2].

Optimising Android Framework

Following measures needs to be taken to optimise the android framework [2]:

- Removing the pre-loading of classes included in zygote(as classes and resources) process reduces the overall boot time considerably.
- Boot time is reduced by customizing the "Launcher.apk" that adds up to the reduction in size of system Image.
- Making a custom boot animation file additionally helps cut back the boot time.
- Package scanning is a time consuming process of the android framework that runs as one thread, scanning all the packages present within the device thus by making multiple threads and scanning these packages in parallel helps cut back the boot up time.

In generalise manner we can say that the modules/supports which are useful for developer and not required by the end user can be removed. There are certain boot time configurations can be loaded afterward. It is advisable that such configuration can be bypassed during booting and can be loaded latter.

IV. Conclusion and Future Scope

As per our study on android boot time optimization, we conclude that in order to reduce Android boot up time, we need to improve NAND throughput and RAM throughput. We can do those at x-loader, u-boot and kernel levels. We also discover that optimizations performed at the Android layer do not yield much reduction in boot up time except for disabling preloaded classes which reduces the boot up time by 5s.Ultimately, the level of boot up time optimization that is achievable is largely subjected to the requirements associated with the platform. While decreasing boot up time we have one side effects of memory usage which is increased as we have decreased Zygote preloading. Also, by changing Package manager as well as Activity Manager, we can reduce the boot up time.

Shortening of boot time has a high scope of future considering any embedded system. As Linux kernel evolves, performance of the system improves with boot time optimization feature. The approach is aimed toward standardizing crucial sections of a whole system's software package stack for achieving quick boot, with specific relation to the android platform. The study and its results is helpful for makers of android based mostly automobile navigation and diversion systems for whom quick boot is extremely essential [5]. In client electronic devices like multimedia players and set high boxes implementation of a quick and economical standby mode is vital. Thus, suspend to RAM and suspend to non-volatile media can form subjects of further research.

References

[1] http://www.androidenea.com/2009/06/android-boot-process-from-power-on.html
[2] https://www.pathpartnertech.com/wp-content/uploads/2014/04/PathPartner_Android_Boot_time-_optimization-_Case-_study-_28_March_14.pdf
[3] http://elinux.org/
[4] K. Baik, C. Jinhee, K. Saena, and W. Suchang. Boosting up Embedded Linux device: experience on Linux-based Smartphone. In Ottawa linux Symposium., 2010
[5] Sridharan Subramanian, "Leveraging Linux to Create an Auto Infotainment Platform" July 2009.
[6] http://www.freescale.com/files/training_pdf/VFTF09_AA114.pdf
[7] G. Singh, K. Bipin, and R. Dhawan. "Optimizing the boot time of android on embedded system," in IEEE 15th International Symposium on Consumer Electronics (ISCE), pages 503 –508, June 2011.
[8] http://processors.wiki.ti.com/index.php/Android_Boot_Time_Optimization.