International Journal of Multimedia and Ubiquitous Engineering
Vol. 12, No. 7 (2017), pp. 19-30
http//dx.doi.org/10.14257/ijmue.2017.12.7.03

Android OS with its Architecture and Android Application with Dalvik Virtual Machine Review

[1]Javed Ahmad Shaheen, [2]Mian Ali Asghar, [3]Abid Hussain

[1]Computer Science Department, Virtual University of Pakistan, Lahore – Pakistan
[2]Computer Science Department, Global Institute Lahore, Lahore – Pakistan
[3]Computer Science Department, Virtual University of Pakistan, Lahore – Pakistan
[1]javedmatyana@gmail.com, [2]aliurooj143@yahoo.com, [3]ms140400004@vu.edu.pk

Abstract

Android OS has broad and open source platform with four layers, commenced with the Android platform and the features of Android applications, gave a detailed picture of Android application framework from the potential of developers. The home screen of devices booted with android have primary navigation and information "hub". These are in Android devices as to the desktop found on personal computers. If we illustrated with a simple music player as example to demonstrate the basic working processes of Android application components as it plays the music by using the service component i.e. media player from class of libraries layers .In this paper, paper could provide guidance to understanding the operation mechanism of Android applications and also give some sense to developing applications on Android platform. This paper also describes some working of Dalvik virtual machine and also elaborates Kernel of Android Operating System.

***Keywords**: Android OS, Android Architecture layers, Android Application, Android Linux Kernel, Android Dalvik virtual machine, Android Application Component.*

1. Introduction

Android applications are mostly developed using Java language by using the Android software Development Kit. Application framework define the common structure of programs in the specific domain. Essentially, an application framework is a component that can be reused; it set the architecture of applications and incorporated as a set of abstract classes and the cooperation of their instances. Android devices are usually powered with battery so Android OS is designed to manage processes to keep power consumption as minimum as it can. In Android devises when an application is not in use the OS shelves its operation while also available if it needs for immediate use rather than closed and during this process it does not use battery power or CPU resources and Android manages the applications stored in memory automatically when memory is low, the system will begin invisible and automatically closing inactive processes, starting with those that have been inactive for longest. [16][17]Life hacker reported in 2011 that third-party task killers were doing more harm than good. Android is an open source operating system based on Linux kernel and launched by Google. Unlike PC operating system, mobile phone operating systems are constrained by their hardware, storage space, power dissipation and mobility conditions. Compared with the development of applications on PC, there are some different features of applications on mobile phone operating systems. This paper introduced the basic architecture and application framework of Android operating system, gives a detailed description of main structure of Android applications and the methods of applications based on Android application framework and Dalvik virtual machine.

2. Introduction of Android OS:

Android is a comprehensive operating environment that is based on Linux® kernel with some version like V2.6, it is also a layered system, the architecture of Android system have shown in this picture, as the picture is showing different layers of android that is Application layer, Application frame work layer, Library layer, Android Run time layer and Kernel layer. Each layer is connected with one another which we will discuss in this paper. Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram

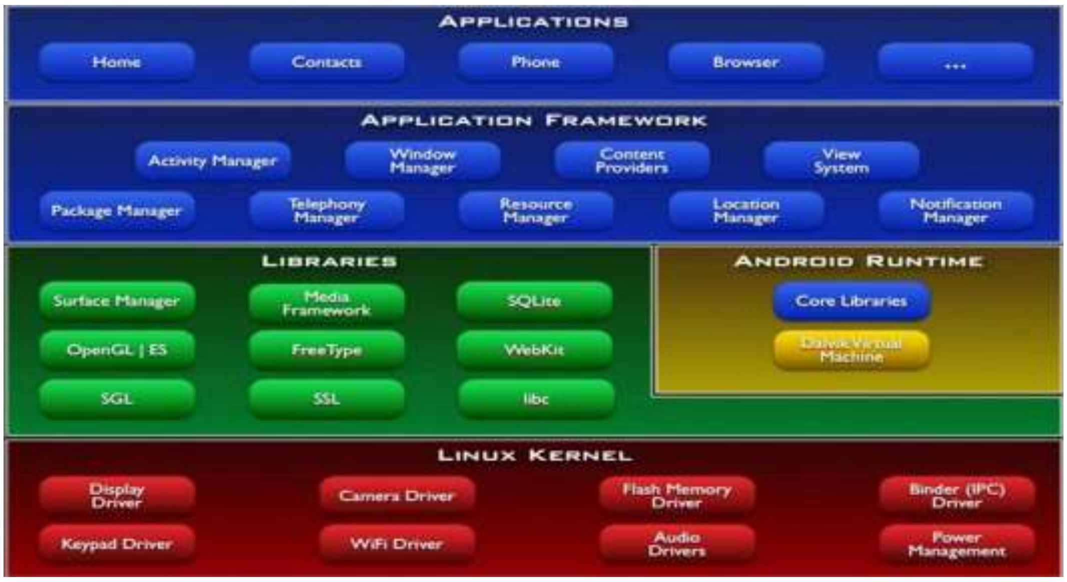

Figure 2.1

2.1 Applications Layer

It is the site of all Android applications which is responsible to include an email client, SMS program, maps, browser, contacts, and others. All the mentioned applications are written using the Java programming language.

2.2 Application Framework Layer

It is the layer which defined the Android application framework. All Android applications based on the application framework. The Android application framework includes:

- A rich and extensible set of **Views** that can be used to build an application with beautiful user interface includes. As it is in our knowledge the view may constituted as lists, grids, text boxes, buttons, and even an embeddable web browser.
- A set of **Content Providers responsible to** enable applications to access data from other applications (such as Contacts), or to share their own data.
- A **Resource Manager responsible to** provides access to noncore resources such as localized strings, graphics, and layout files.
- A **Notification Manager responsible to** enables all applications to display custom alerts in the status bar.
- An **Activity Manager is responsible to** manage the lifecycle of applications and provides a common navigation back stack.

- **Location manager:** It is **responsible to** fires alerts when user enters or leaves a specified geographical location.
- **Package manager:** It is **responsible to** retrieve the data about installed packages on device.
- **Window manager:** It is **responsible to** create views and layouts.
- **Telephony manager:** It is **responsible to** handle settings of network connection and all information about services on device.[1]

2.3. Application Library

GNU libs (glibc) is too big and complicated for mobile phones, so Android implements its own special version of libc, *Bionic libc which is in* Smaller size - 200K in it some of features strip out some complicated C++ features, the most significant one no C++ exception, Very special and small thread implementation, heavily based on kernel futexes. Bionic libc does *not* fully support POSIX and is *not* compatible with glibc.

Libraries layer includes a set of C/C++ libraries used by various components of the Android system and provides support to the application framework

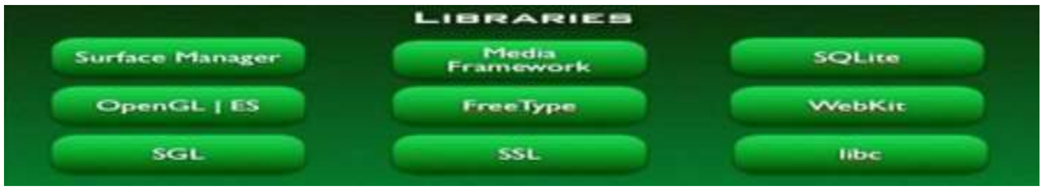

Figure 2.3

2.4 Android Runtime

It includes a set of core libraries and a Java virtual machine (Dalvik virtual machine) that has been redesigned and optimized by Google to be suitable for Android platform. Linux kernel is located at bottom layer of Android system and acts as an abstraction layer between the hardware and the rest of the software stack. It provides core system services such as security, memory management, process management, network stack, and driver model. In addition, some bottom functions such as management of threads of Dalvik virtual machine also rely on the Linux kernel.

2.4.1 Android Application Runtime Environment

Each Android application runs in a separate process, with its own instance of the Dalvik virtual machine (VM). Based on the Java VM, the Dalvik design has been optimized for mobile devices. The Dalvik VM has a small memory footprint and multiple instances of the Dalvik VM can run concurrently on the handset

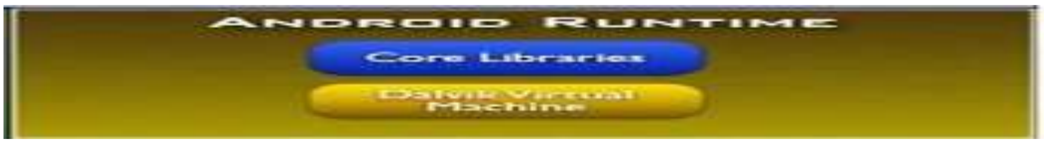

Figure 2.4

2.5 Linux Kernel

At the bottom of the layers is Linux. Linux 2.6 with approximately 115 patches. This provides basic system functionality like process management, memory management, device management like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

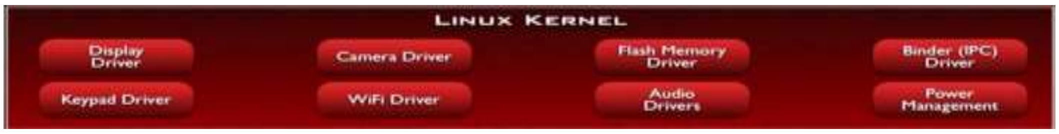

Figure 2.5

3. Dalvik Virtual Machine

Android applications and the underlying frameworks are almost entirely written in Java. Instead of using a standard Java virtual machine, Android uses its own VM. This virtual machine is not compatible to the standard Java virtual machine Java ME as it is specialized and optimized for small systems. These small systems usually only provide little RAM, a slow CPU and other than most PCs no swap space to compensate the small amount of memory Android is running on the Linux kernel and its applications are written by Java programming language, so Android applications are running on a Java virtual machine named Dalvik virtual machine. Dalvik virtual machine has been redesigned and optimized by Google for the hardware features of mobile devices. The necessary byte code interpreter the virtual machine is called Dalvik. Instead of using standard byte code, Dalvik has it's own byte code format which is adjusted to the needs of Android target devices. The byte code is more compact than usual Java byte code and the generated .dex files are small. In Android system, there is a tool named .dex, included in the Android SDK, transforms the Java Class files (which compiled by a regular Java compiler) into the .dex format. The .dex format files integrate all Java class files and delete redundant information in every Java class files.

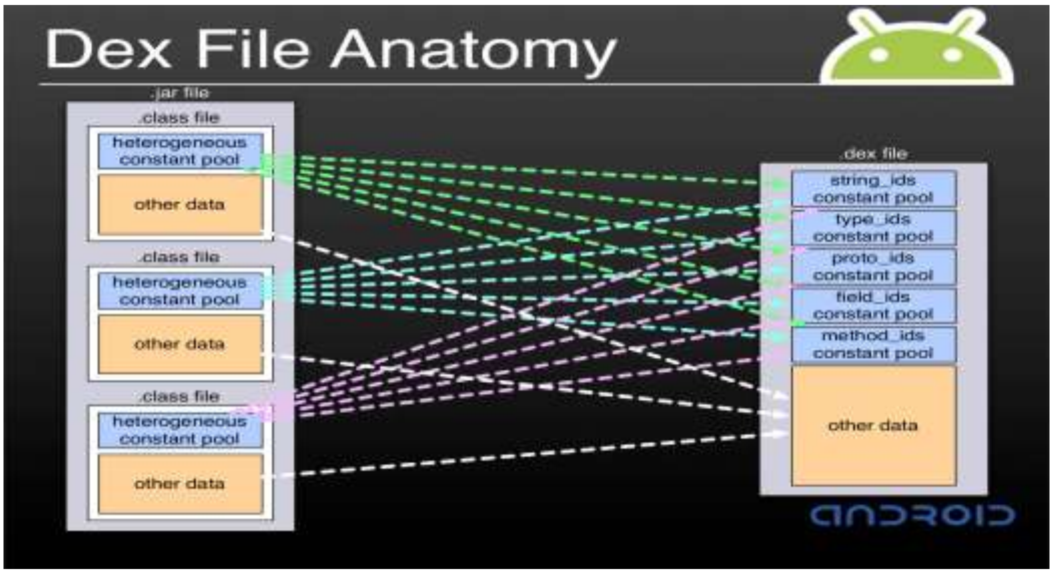

Figure 3. Showing .dex File Format

3.1. Memory in DVM

In Dalvik VM there are four different kinds of memory to distinguish that can be grouped to **clean/dirty** and **shared/private**. Typical data residing in either shared or private clean memory are libraries and application specific files like .dex files. Clean memory is backed up by files or other sources and can be pruned by the kernel without data loss. The private dirty memory usually consists of the applications heap and writeable control data structures like those needed in .dex files. These three categories of different memory are quite common and no specialty of Dalvik.

3.2. Feature of DVM

There are several features of Dalvik virtual machine:

3.2.1 **Zygote**:[12]

Shared dirty memory is possible through a facility of Dalvik called Zygote. It is a process which starts at boot time and is the parent of all Dalvik VMs in the system. The Zygote loads and initializes classes that are supposed to be used very often by applications into its heap. In shared dirty memory resides e.g. the dex data structure of libraries. After the startup of the Zygote, it listens to commands on a socket. If a new application starts, a command is sent to the Zygote which performs a standard fork (). The newly forked process becomes a full Dalvik VM running the started application. The shared dirty memory is “copy-on-write” memory to minimize the memory consumption

- Dalvik virtual machine has multiple occurrence / instances on one device and every instance runs in a separate Linux process, an Android application runs in an instance of a Dalvik virtual machine.
- Dalvik virtual machine relies on the underlying operating system (Linux kernel) for process isolation, memory management and threading support.
- Dalvik virtual machine is register based.

The follow figure (figure3-2) shows the position of Dalvik virtual machine in Android system.

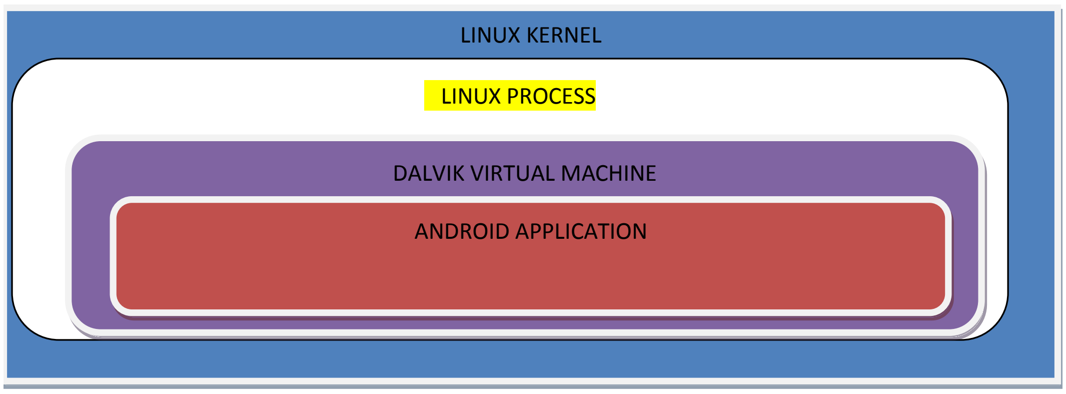

Figure 3.2. DVM Position in Android

4. Android Application Component

A core feature of Android is that one application could use component element that belong to another application mean if the component is permitted using. In order to achieve such functions, Android system must launch the application while any part of the application is asked and instantiate Java objects that being asked. Unlike most operating system, there is no single point that the system can enter in an Android application (for example, there no main ( ) function in an Android application). Instead, each component is a different point through which the system can enter an application and instantiate component object independently.

There are four different types of application components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

4.1 Activity

An activity represents a single screen with a user interface. The activities in an application work together to form a cohesive user experience, but each one is independent of the others. As such, a different application can start any one of these activities. An activity is implemented as a subclass of Activity. The particular form that an activity show users and the amount of activities in an application depend on how the developer design the application. In a multiple activities application, typically, one activity is specified as the "main" activity, which is presented to the user when launching the application for the first time. Each activity can then start another activity in order to perform different actions. Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack the back stack [1]

4.2 Service

A service is an Android component that runs in the background to perform long-running operations or to perform work for remote processes and does not provide a user interface. An activity can connect or bind a service that is running. If the service is not running, launch it. When connected to a service, the activity can communicate with the service through the interface that the service exposed. Like other application components, service components always running in the main thread of an application by default. So for the intensive or blocking operating a service performs may slow down activity performance, it is usually start a new thread inside the service that is content provider,

4.3 Content Providers

Content providers provide data share mechanism among applications. The data that be shared could be in the file system, a SQLite database, or any other persistent storage location an application can access. A content provider is implemented as a subclass of Content provider, it defines the data format and it supported and provides a set of method to enable other applications to query or modify the data. But an application does not call these methods immediately, instead it call these methods by an object named content resolver.

4.4 Content Resolver

Content Resolver can communicate with every Content Provider. Content Resolver cooperated with Content Provider to manger IPC (inter process communication) while sharing data.

4.5. Broadcast Receivers

Broadcast Receivers is in charge of the reception of system wide broadcast and take response aiming at the information that a broadcast transmitted. Many broadcasts originate from the system for example, a broadcast announcing that the screen has turned off, the battery is low. Applications can also initiate broadcasts. There could be any number of Broadcast Receivers in an application and each Broadcast Receiver implemented as a sub class of Broadcast Receiver. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work.[10]

5. Intent

Three of the four Application component types i.e. activities, services, and broadcast receivers are activated by an asynchronous message called intent. Intents bind individual components to each other at runtime no matter the component belongs to the same application. Intent can create with an Intent object, which defines the messages by which can activate either a specific component or a specific type of component.

5.1. Intent Action

This is mandatory part of the Intent object and is a string naming the action to be performed or, in the case of broadcast intents, the action that took place and is being reported. The action largely determines how the rest of the intent object is structured. The Intent class defines a number of action constants corresponding to different intents intent defines the action to perform and may specify the URI of the data to act on for broadcast receivers; the intent simply defines the announcement being broadcast. The other component type, content provider, is not activated by intents. Rather, it is activated when targeted by a request from a Content Resolver.[3].

5.1.2. Android Intent Standard Actions:

Following table lists down some of important Android Intent Standard Actions. You can check Android Official Documentation for a complete list of Actions:

Table 5.1

Sr. No	Intent Action and description
1	ACTION_ALL_APPS: **List all the applications available on the device.**
2	ACTION_ANSWER: **Handle an incoming phone call.**
3	ACTION_ATTACH_DATA: **Used to indicate that some piece of data should be attached to some other place**
4	ACTION_BATTERY_CHANGED: **This is a sticky broadcast containing the charging state, level, and other information about the battery.**
5	ACTION_BATTERY_LOW: **This broadcast corresponds to the "Low battery warning" system dialog.**
6	ACTION_BATTERY_OKAY: **This will be sent after ACTION_BATTERY_LOW once the battery has gone back up to an okay state.**
7	ACTION_BOOT_COMPLETED : **This is broadcast once, after the system has finished booting.**
8	ACTION_BUG_REPORT: **Show activity for reporting a bug.**
9	ACTION_CALL : **Perform a call to someone specified by the data.**
10	ACTION_CALL_BUTTON: **The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call.**
11	ACTION_CAMERA_BUTTON: **The "Camera Button" was pressed.**
12	ACTION_CHOOSER: **Display an activity chooser, allowing the user to pick what they want to before proceeding.**
13	ACTION_CONFIGURATION_CHANGED: **The current device Configuration (orientation, locale, etc) has changed.**
14	ACTION_DATE_CHANGED: **The date has changed.**
15	ACTION_DEFAULT: **A synonym for ACTION_VIEW, the "standard" action that is performed on a piece of data.**

16	**ACTION_DELETE: Delete the given data from its container.**
17	**ACTION_DEVICE_STORAGE_LOW: A sticky broadcast that indicates low memory condition on the device.**
18	**ACTION_DEVICE_STORAGE_OK: Indicates low memory condition on the device no longer exists.**
19	**ACTION_DIAL: Dial a number as specified by the data.**
20	**ACTION_DOCK_EVENT A sticky broadcast for changes in the physical docking state of the device.**
21	**ACTION_DREAMING_STARTED Sent after the system starts dreaming.**
22	**ACTION_DREAMING_STOPPED Sent after the system stops dreaming.**
23	**ACTION_EDIT Provide explicit editable access to the given data.**
24	**ACTION_FACTORY_TEST Main entry point for factory tests.**
25	**ACTION_GET_CONTENT Allow the user to select a particular kind of data and return it.**
26	**ACTION_GTALK_SERVICE_CONNECTED A GTalk connection has been established.**
27	**ACTION_GTALK_SERVICE_DISCONNECTED A GTalk connection has been disconnected.**
28	**ACTION_HEADSET_PLUG Wired Headset plugged in or unplugged.**
29	**ACTION_INPUT_METHOD_CHANGED An input method has been changed.**
30	**ACTION_INSERT Insert an empty item into the given container.**
31	**ACTION_INSERT_OR_EDIT Pick an existing item, or insert a new item, and then edit it.**
32	**ACTION_INSTALL_PACKAGE Launch application installer.**
33	**ACTION_LOCALE_CHANGED The current device's locale has changed.**
34	**ACTION_MAIN Start as a main entry point, does not expect to receive data.**
35	**ACTION_MEDIA_BUTTON The "Media Button" was pressed.**

6. Boot Sequence and Process

Android boot sequence is hereby illustrated in the diagram 6.1, which describes that whenever application want to run, it is connected with in application framework with its specific manager and then it is jointed with native libraries to its native library and then in run time Dalvik Virtual machine which is loaded by zygote through kernel and it is initiated that application related class in kernel.

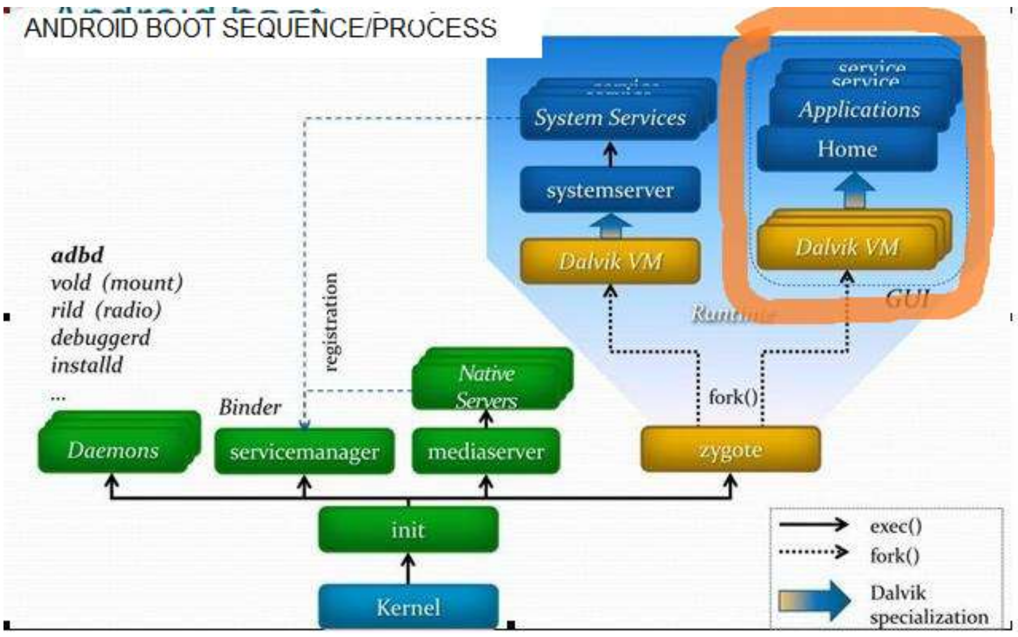

Figure 6.1 Android Boot Sequence

6.1. AN MUSIC PLAYER AS Example for Boot Sequence

Here there is a simple music player; the four components of Android have been defined in this example. Music Main Activity is an object of Activity type, it provides interface to users in Application layer and communicates with a Service used to play music in background by a Broadcast Receiver. Music Player Service is an object of extensional service type, it's mainly function is to play music in background and return play statues to Music Main Activity by broadcast. MusicInfo Manager is a custom class; it packs a Content Providers provided by system to get music information from flash card. By the above components cooperate with each other can realize the function of play music on android platform. The MusicPlayerService run in background returns all play status to the MusicMainActivity run in foreground by broadcast. A Broadcast Receiver has registered in MusicMainActivity to receive all broadcast from MusicPlayerService. The received broadcast will be resolved by MusicMainActivity. According to the broadcast content, the MusicMainActivity will do some actions shows users the play status. After the music player launched, the MusicMainActivity will send messages to MusicInfoManager for the information of music files and the MusicInfoManager will activate a ContentProvider that provided by system, then get the music files list and return it to the MusicMainActivity.

6.2. Android Boot Process:

In figure 6.2 we can see the boot process of android when it is switched on and the detail of these processes is under"

Step 1. Power On and System Startup: When power starts, Boot ROM code start execution from pre-defined location i.e. is hardwired on ROM. It load Bootloader into RAM and start execution.

Step 2. Bootloader: Bootloader is small program which runs before Android operating system running. Bootloader is first program to run so It is specific for board and processor.
Bootloader perform execution in two stages, first stage It to detect external RAM and load program which helps in second stage, In second stage bootloader setup network, memory, etc. which requires to run kernel, bootloader is able to provide configuration parameters or inputs to the kernel for specific purpose.

Step 3: Kernel: Android kernel start similar way as desktop Linux kernel starts, as kernel launch it start setup cache, protected memory, scheduling, loads drivers. When kernel finishes system setup first thing it look for "init" in system files and launch root process or first process of system.

Step 4: init process: it very first process, we can say it is root process or grandmother of all processes. Init process has two responsibilities
1. Mount directories like /sys, /dev, /proc
2. Run init.rc script.

Step 5: Zygote and Dalvik: We know that separate Virtual Machine (VMs) instance will pop up in memory for separate per app in JAVA but in Android app should launch quick as possible. If Android OS launch different instance of Dalvik VM for every app then it consume lots of memory and time. So, to overcome this problem Android OS has "Zygote". Zygote enable shared code across Dalvik VM, lower memory footprint and minimal startup time. Zygote is a VM process that starts at system boot time as we discussed in previous step. Zygote preloads and initializes core library classes. These are three core classes which are read-only and part of Android SDK or Core frameworks.

Step 6. System Service: After completion of all steps runtime request Zygote to launch system servers. These are written in both native and java. The System servers in JAVA can consider as process as the same system server is available as System Services in Android SDK. Zygote fork new process to launch system services and we can see source code in ZygoteInit class and "startSystemServer" method.

Step 7 : Boot Completed: As System Services starts up and running in memory, Android has completed booting process and "ACTION_BOOT_COMPLETED" standard broadcast action will fire.

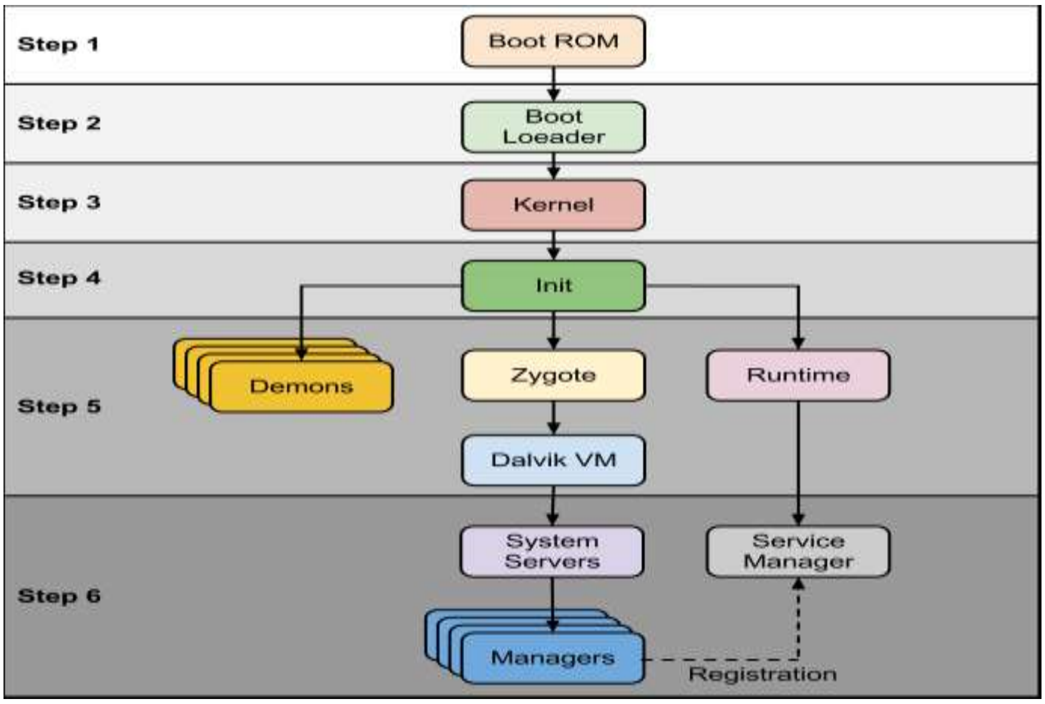

Figure 6.2

7. Conclusion

Android as a full, open and free mobile device platform with its powerful function and good user experience rapidly developed into the most popular mobile operating system. This article gives a detailed introduction of Android OS, Android layers application framework and native libraries and the working principal of Android applications run time and Dalvik virtual machine. As a final point, a music player on the android platform has presented as an example to point up this mechanism how boot sequence of Android carry on and this paper also have brief knowledge about Android Components.

References

[1] OL. Google Android Developers, Android Develop Guide,http://developer.android.com/guide/topics/fundamentals.html
[2] J. Liu, J. Yu, "Research on Development of Android Applications", Fourth International Conference on Intelligent Networks and Intelligent Systems, **(2011)**.
[3] M. F. Yang, "Android Application Development Revelation", China Machine Press, vol. 11, **(2010)**.
[4] M. Zhengguo Hu, Jian Wu, Zhenggong Deng, Programming Methodology, National Defence Industry Press, vol. 6, **(2008)**.
[5] M. Junmin Ye, "Software Engineering", Tsinghua University Press, vol.6, **(2006)**.
[6] tutorialspoint.com
[7] J. Dongjiu Geng, Yue Suo, Yu Chen, Jun Wen, Yongqing Lu, "Remote Access and Control System Based on Android Mobil Phone", Journal of Computer Applications, vol.2, **(2011)**, pp. 560-562.

[8] J. Li Lin, Changwi Zou, "Research on Cloud Computing Based on Android Platform", Software Guide, vol.11, **(2010)**, pp.137-139
[9] http://en.wikipedia.org/wiki/Android_(operating_system)
[10] http://www.openhandsetalliance.com/android_overview.html
[11] http://www.android.com
[12] D. Bornstein, "Dalvik VM Internals", Google I/O conference 2008 presentation video and slides. http://sites.google.com/site/ io/Dalvik-vm-internals, **(2008)**.
[13] B. Dolan-Gavitt, *et al*, "Virtuoso: Narrowing the Semantic Gap in Virtual Machine Introspection", IEEE Security and Privacy, **(2011)**.
[14] "The truth about Android task killers and why you don't need them". PhoneDog, **(2011)**, Retrieved October 30, 2012.
[15] V. Matos "Lesson 3: Android Application's Life Cycle" (PDF). grail.cba.csuohio.edu. Cleveland State University. Archived from the original (PDF), **(2014)**. Retrieved April 15, 2014.
[16] "Android PSA: Stop Using Task Killer Apps". Phandroid.com. **(2011)**. Retrieved October 30, 2012.
[17] R. Meier. Professional Android 4 Application Development. Books.google.com. Retrieved February 9, 2014.

Authors

Javed Ahmad Shaheen, He is presently working as SST (CS) in Punjab School Education Department; he has done his MSCS (Networking) from Virtual University of Pakistan Lahore. He has practical experience of Network installation.

Mian Ali Asghar, He is also presently working as SST (CS) in Punjab School Education Department; he is doing his MSCS from Global Institute Lahore. He has good command over Web Development and Software Engineering.

Abid Hussain, He is also presently working as SST (CS) in Punjab School Education Department; he is doing his MSCS (Networking) from Virtual University of Pakistan Lahore. He has good command over Wireless Networking and Programming.