CECSC09 - Operating Systems

**Case Study on Android**

Aaryan Raj Sarda (2019UCO1684)

Vijay Pal (2019UCO1676)

*NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY*

# 

# 

# 

# Table Of Contents

[**Table Of Contents**](#_fimnlvt541u) **2**

[**Introduction**](#_b0xru08t0veg) **3**

[**Architecture**](#_6m308lsq8fcj) **3**

[Applications Layer](#_v05byu6a1e48) 4

[Application Framework Layer](#_h6hjmwnhnuxv) 4

[Android Runtime](#_xb3416kk48ix) 5

[Libraries](#_wxgwp3dkjimr) 6

[Linux Kernel](#_chvelgio3vph) 6

[**Process and Threads**](#_9icyf1nmdne) **7**

[Android Applications](#_b1r6fvtxd69s) 7

[Android Booting Process](#_7yofjfk9nx8) 7

[Attributes of a Process](#_w22aj0oqbupk) 9

[The Priority of Processes In Android Application](#_q5ytvahx69xy) 12

[Inter Process Communication](#_c6w2mg62uq81) 13

[Process Scheduling](#_sz5w7w75gzoc) 14

[Process Termination](#_n1ts6kacxklj) 14

[**Threads**](#_s5h4d2lhrlpe) **15**

[Process Synchronisation](#_w9vlcsivmqfk) 16

[**Conclusion**](#_11bnn6gcb4ra) **16**

[**References**](#_11lrkhbctajp) **18**

# Introduction

Android is intended to revolutionize the mobile market by bringing the internet to the cell phone and allowing its use in the same way as on the PC. The term “Android” has its origin in the Greek word andr-, meaning “man or male” and the suffix - eides, used to mean “alike or of the species”. This together means as much as “being human”.

As smart phones and tablets become more popular, the operating systems for those devices become more important. Android is such an operating system for low powered devices that run on battery and are full of hardware like Global Positioning System (GPS) receivers, cameras, light and orientation sensors, Wi-Fi and UMTS (3G telephony) connectivity and a touch screen. Like all operating systems, Android enables applications to make use of the hardware features through abstraction and provide a defined environment for applications.

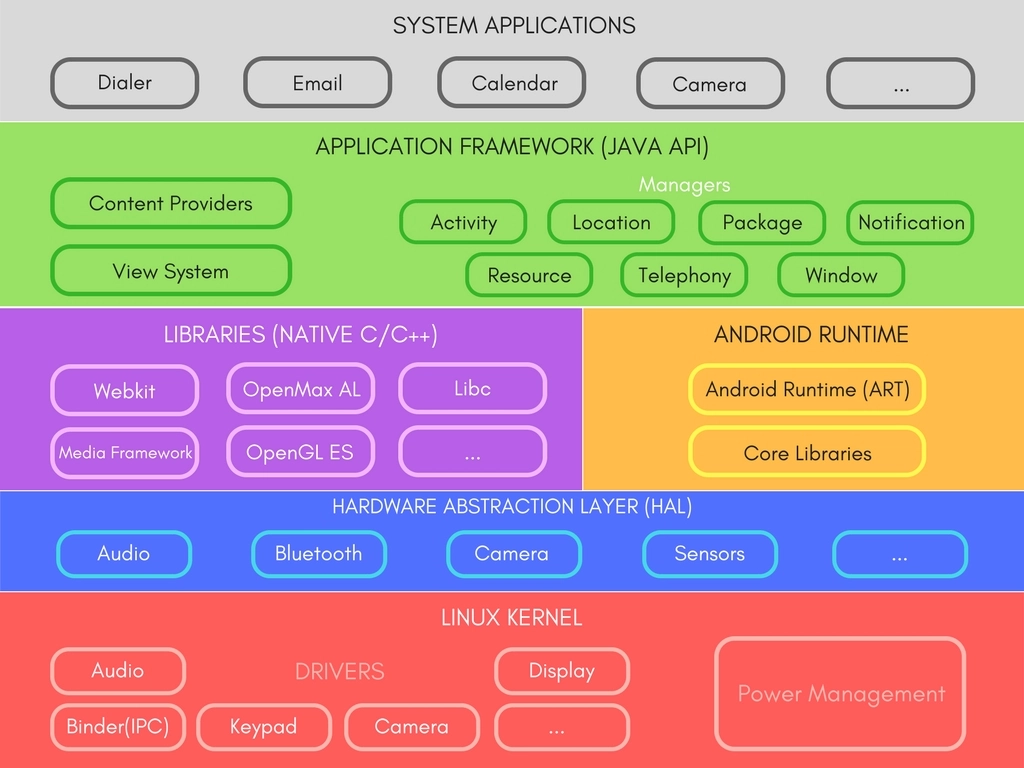
Android is described as a mobile operating system, initially developed by Android Inc. Android was sold to Google in 2005. Android is based on a modified Linux 2.6 kernel. Google, as well as other members of the Open Handset Alliance (OHA) collaborated on Android (design, development, distribution). Currently, the Android Open Source Project (AOSP) is governing the Android maintenance and development cycle.

The development of Android takes place quickly, as a new major release happens every few months. This leads to a situation where information about the platform becomes obsolete very quickly and sources like books and articles can hardly keep up with the development. Sources that keep up with the pace are foremost the extensive SDK documentation, documentation in and the source code itself as well as blogs.

This report discusses the major components that comprise the Android operating environment, elaborating on the Android design and architecture (the building blocks).

# Architecture

Android is a comprehensive operating environment that is based on Linux kernel, it is also a layered system; the architecture of Android system is shown as in picture.



## Applications Layer

Application Layer is the site of all Android applications including an email client, SMS program, maps, browser, contacts, and others. All applications are written using the Java programming language.

## Application Framework Layer

Application Framework Layer defines the Android application framework. All Android applications are based on the application framework. The Android application framework includes:

1. A rich and extensible set of **Views** that can be used to build an application with a beautiful user interface, including lists, grids, text boxes, buttons, and even an embeddable web browser.
2. A set of **Content Providers** that enable applications to access data from other applications (such as Contacts), or to share their own data.
3. A **Resource Manager** that provides access to non code resources such as localized strings, graphics, and layout files.
4. A **Notification Manager** that enables all applications to display custom alerts in the status bar.
5. An **Activity Manager** that manages the lifecycle of applications and provides a common navigation back stack. It enables proper management of all the activities. All the activities are controlled by the activity manager.
6. A **Location Manager** that fires alerts when a user enters or leaves a specified geographical location.
7. A **Package Manager** is used to retrieve the data about installed packages on a device.
8. A **Window Manager** is used to create views and layouts.
9. A **Telephony Manager** is used to handle settings of network connection and all information about services on device.

## Android Runtime

This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called **Dalvik Virtual Machine** which is a kind of Java Virtual Machine specially designed and optimized for Android.

The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. T**he Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.**

The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

## Libraries

Android has its own libraries, which is written in C/C++. These libraries cannot be accessed directly. With the help of the application framework, we can access these libraries. There are many libraries like web libraries to access web browsers, libraries for android and video formats etc.

1. **Media** library provides support to play and record audio and video formats.
2. **Surface manager** is responsible for managing access to the display subsystem.
3. **SGL** and **OpenGL** both cross-language, cross-platform application program interfaces (API) are used for 2D and 3D computer graphics.
4. **SQLite** provides database support and **FreeType** provides font support.
5. **Web-Kit** is an open source web browser engine that provides all the functionality to display web content and to simplify page loading.
6. **SSL (Secure Sockets Layer)** is a security technology to establish an encrypted link between a web server and a web browser.

## Linux Kernel

This layer is the core of android architecture. It helps in software or hardware binding for better communication. It manages all the available drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc. which are required during the runtime. The Linux Kernel will provide an abstraction layer between the device hardware and the other components of android architecture. It is responsible for management of memory, power, devices etc. The features of Linux kernel are:

1. **Security:** The Linux kernel handles the security between the application and the system.
2. **Memory Management:** It efficiently handles the memory management thereby providing the freedom to develop our apps.
3. **Process Management:** It manages the process well, allocates resources to processes whenever they need them.
4. **Network Stack:** It effectively handles the network communication.
5. **Driver Model:** It ensures that the application works properly on the device and hardware manufacturers responsible for building their drivers into the Linux build.

# Process and Threads

A process in an operating system is the program in active state. So process management is a way by which different processes share resources and execute in an optimal way so that the properties like synchronisation, mutual exclusion are maintained.

Android process management is similar to that of Linux at a low level, but the Android Runtime provides a layer of abstraction to help keep often used processes in memory as long as it can. This is done using some memory management techniques that are not common.

## Android Applications

Android applications differ from standard applications in a couple very significant ways.

* Every android application runs in a separate process, has its own Dalvik VM and since Android is a single user OS, the designers assign each application a unique UID at install time. This means the underlying Linux kernel can protect each application's files and memory without additional effort.
* There is no single entry point for android applications. An application is a collection of components that can be used in other applications if desired.

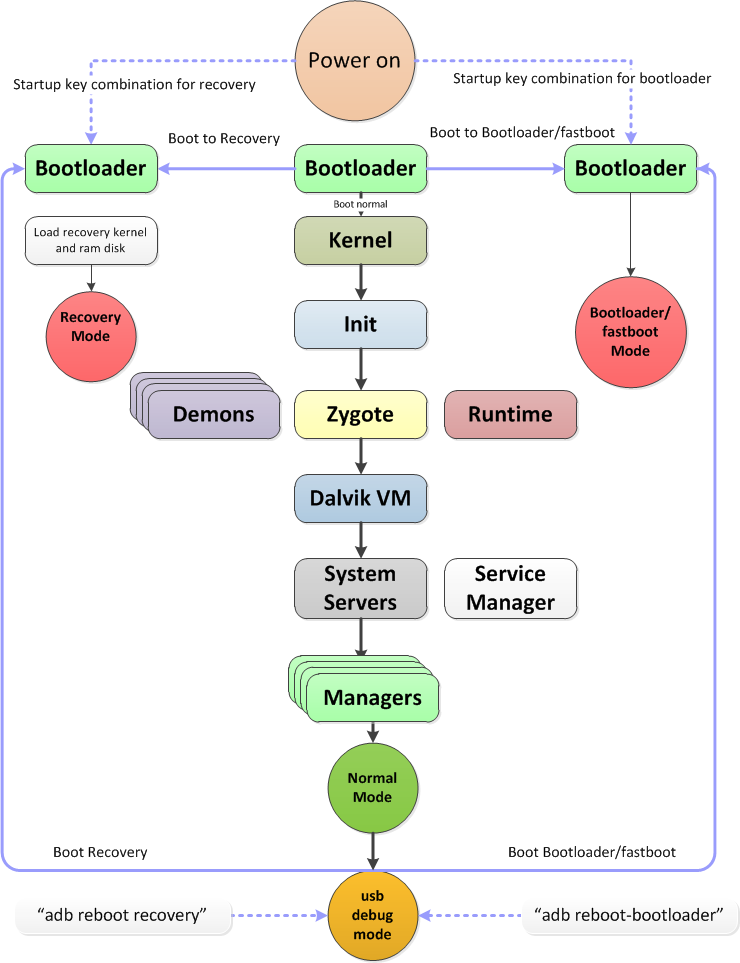
## Android Booting Process

Android Boot Process is more or less similar to the Linux booting process. We all know that android is an application running on the top of Linux operating systems. More popularly known as "A Stack Of Application".

The Android process has the following sequence

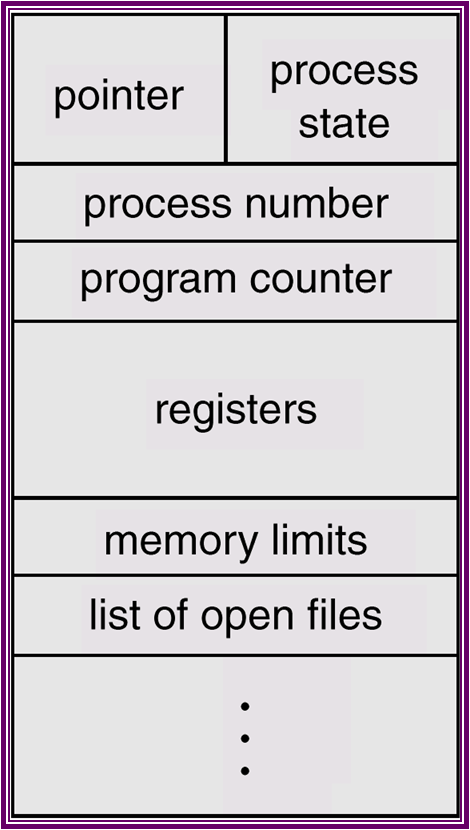
* Init
* Zygote
* System Server
* Service Manager
* Other Daemons and processes
* Applications

The Pictorial Representation is as follows:

****

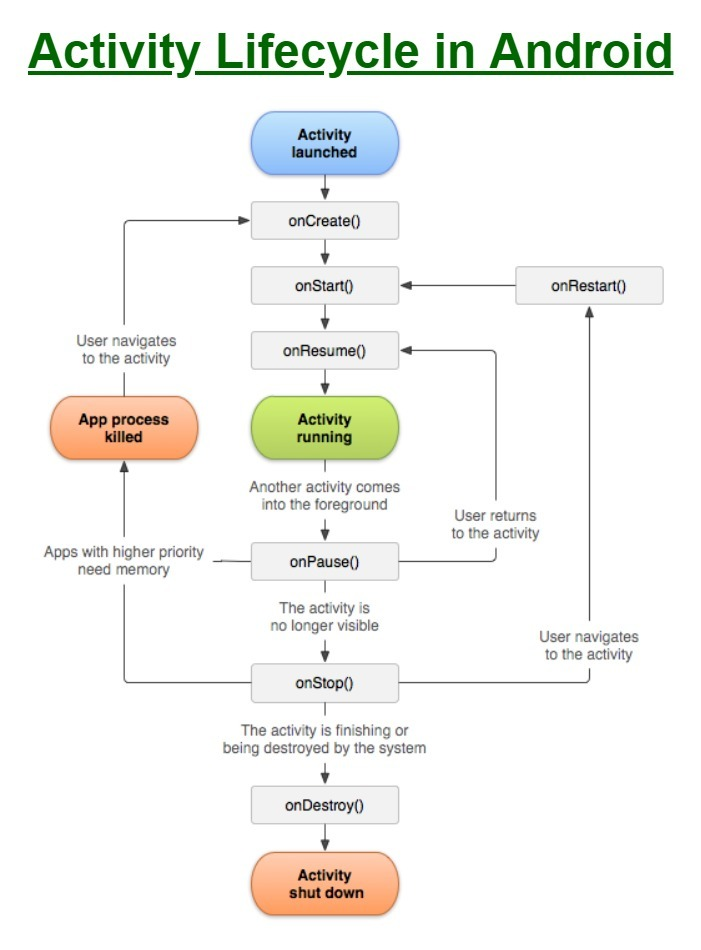
## Attributes of a Process

Process management in a typical operating system involves many complex data structures and algorithms, but doesn’t go much beyond the level managing the typical process data structure. Android is similar in that at the base level, the control structures look the same. The Process Control Block looks like :



Now in Android, each application consists of a process in which several activities run so it is important to study the life cycle of an activity in order to study the life cycle of a process. An activity is basically a single screen with which the user interacts. The activities are managed as an activity stack. The android OS uses priority queue for the management of the activities running on the device. The priority system helps Android identify activities that are no longer available so that the android OS can reclaim the resources and memory from those activities. There are four states of an activity:

* **Active or running**: An activity is said to be in active state if it is running in the foreground, also known as the top of the activity stack. These activities have the highest priorities and can be killed only in extreme situations i.e. when the activity requires more memory than the memory available on the device.
* **Paused**: If an activity has lost focus but is still visible then it is in the pause state, a paused activity is alive, but can be killed by the system in an extreme low memory situation by the OOM.
* **Stopped**: The activity is completely stopped either the execution has been completed or the OS requires memory to execute another process.
* **Destroyed**: After the activity has been stopped it is destroyed so as to get all the resources that were previously used by the activity, it is a very useful step because android is an OS with low memory and also low computation power.



Now there are several types of limitation in android os due to its small size. So it is important to develop a mechanism for the management of processes. That's why a Process hierarchy is developed so that processes with less priority must be killed.

## The Priority of Processes In Android Application

To determine which process should be killed to proper memory and battery management, the Android operating system maintains a hierarchy in which all the processes are placed in order of their priority. The less priority is the process which can be killed first when the system wants to free up some resource. Android uses some set of rules and regulations to decide the priority of the processes based on the running state of the applications. Below are the process states that a process may have at any time in android applications. **The priority of these processes decreases from top to down in order in which they are listed.**

1. **Foreground process**

A foreground process is a process with which the user is currently interacting and using it. A process is considered to be in the foreground state if any of the below conditions hold:

* If the process is running an activity with which the user is interacting
* If it has a broadcast receiver which is currently in execution to receive any system update.

1. **Visible process**

A visible process is a process when the activity can be visible to the user. The user does not directly interact with this process, as the activity corresponding to this process would be covered partially by another activity and the process will be in the onPause() lifecycle state. This process cannot be killed unless there is so much lack of memory in the system that the execution of these processes cannot be possible. Killing these processes will create a negative impact on user experience, as a user can see the activity corresponding to this process. These processes would be killed only when keeping them alive would make it impossible for the foreground process to continue their execution.

1. **Service Process**

A process is said to be a service process if it is in running state and neither a foreground process and a visible process. These processes are not directly visible to the user of the application. This process is helpful for the applications which perform the background tasks such as background network data upload or download. The system will keep the service process alive until it becomes impossible for the system to keep the foreground process and visible process running.

1. **Background Process**

A background state in which the onStop() lifecycle method of android is called by the system. Let’s suppose the user is using an app and suddenly presses the home button, so because of this action, the process goes from foreground state to background state. When the app goes from foreground state to background state, it goes to the LRU cache queue and will be placed in the front of the queue.when the user returns to that app, the process will return from background state to foreground state. So having knowledge of the process and application lifecycle in android along with how processes can decide the life cycle time of the application is a must for an android developer which can lead to good user experience.

## Inter Process Communication

IPC is inter-process communication. It describes the mechanisms used by different types of android components to communicate with one another.

1. **Intents** are messages which components can send and receive. It is a universal mechanism of passing data between processes. With help of the intents one can start services or activities, invoke broadcast receivers and so on.
2. **Bundles** are entities of data that are passed through. It is similar to the serialization of an object, but much faster on android. Bundle can be read from intent via the getExtras() method.
3. **Binders** are the entities which allow activities and services to obtain a reference to another service. It allows not simply sending messages to services but directly invoking methods on them.

## Process Scheduling

Similar to other operating systems the android also uses the process scheduling techniques. It supports all the scheduling techniques like first in first out, round robin scheduling and scheduling techniques so as to execute the processes in an efficient manner. Now in order to perform the scheduling techniques different types of scheduler are used, for e.g. some I/O scheduler are FIFO (first in first out), CFQ (complete fair Queue that divides the available I/O bandwidth equally), SIO (simple input output to keep minimum no of overheads) etc.

## Process Termination

Processes can be killed in a couple discrete ways.

1. An application can call a [method](http://developer.android.com/reference/android/os/Process.html#killProcess(int)) to kill processes it has permission to kill. This means if the process isn't part of the same application, it can't kill other processes. On install you can actually grant an application permission to kill other applications, but this is something you don't typically do.
2. The Android OS has a least recently used queue that keeps track of which applications haven't been used. If the OS starts to run out of memory, it will kill the least recently used application. There is also priority given to applications that a user is interacting with, or background services the user is interacting with.

A process can be killed using **Process.killProcess(int, pid)** where pid is the process id of the process to be killed.

## 

## 

## 

## Threads

Thread is a basic unit of CPU execution. Thread has a thread ID, program counter, a register set, and a stack similar to a process.

The default thread that creates UI is known as UI thread or main thread. All other threads are known as Worker threads.

When an application is launched, the system creates a main/UI thread of execution for the application. This thread is in charge of dispatching events to the appropriate user interface widgets. It is also almost always the thread in which your application interacts with Android UI. As such, the main thread is also sometimes called the UI thread.

The system does not create a separate thread for each instance of a component. All components that run in the same process are instantiated in the UI thread or main thread, and system calls to each component are dispatched from that thread.

For instance, when the user touches a button on the screen, your app’s UI thread dispatches the touch event to the widget, which in turn sets its pressed state and posts an invalidate request to the event queue. The UI thread dequeues the request and notifies the widget that it should redraw itself.

When your app performs intensive work in response to user interaction, this single thread execution results in poor performance.

Specifically, if everything is happening in the UI thread, performing long operations such as network access or database queries will block the whole UI. When the thread is blocked, no events can be dispatched, including drawing events. From the user’s perspective, the application appears to hang.

Additionally, the Android UI toolkit is not thread-safe. So, you must not manipulate your UI from a worker thread—you must do all manipulation to your user interface from the UI thread. Thus, there are simply two rules to Android’s single thread model:

* Do not block the UI thread.
* Do not access the Android UI from outside the UI thread.

## Process Synchronisation

There are sometimes situations when two or more processes want to share a resource so in order to avoid the conflicts between the processes locking mechanism is used in android so as to ensure the one resource is available to a single process at a time. These are generally implemented using the semaphores which is a variable that is assigned a particular value and depending on these values locks are granted and permission are taken back form the processes, also in android we generally use semaphores with only two values i.e. 0 and 1 also called as mutex . At the high level these are implemented using the *java.util.concurrent.locks.Lock* class.

# Conclusion

In these days, Android has become a very popular operating system for smartphones. There are some advanced features in android Smartphone, with which user can easily share applications via online market store i.e. Google market store. But, there are attacks and threats included in this platform, like malware applications are also attacks on Android actual applications. Because malware on devices can create a number of risks, which creates problems while connectivity because of security issues. In this paper, it will be described how security can be improved on the Android Operating System so that users can safely use the android smartphones.

Elaborating on the major components that comprise the Android operating environment, this report focused on providing a comprehensive overview of the status quo. The very impressive, rapid evolution of Android resembles the great work done by the Linux community over the years. As discussed in this report, the android architecture has been discussed in a detailed way and identified several security issues in using android devices and found some solutions for overcoming the security issues for both users and developers.

With such a rapidly developing environment, both in terms of product innovation and the threat landscape, other security considerations will rapidly develop in the months and years to come. The measures discussed in this text serve as a good starting point in providing a baseline of security on Android devices. The preferable solution would theoretically be not to allow personal devices on to the network at all, and this may prove an effective if sometimes unpopular decision. The risk-reward ratio is never going to be appealing to a security professional, however this is one of the lesser concerns amongst users. There is no one-stop effective security measure that can be implemented on an Android device. Certainly when it comes to corporate devices then one of the emerging products provides some much needed functionality to the mobile security tool kit. These solutions however are difficult for organizations to implement on personal devices, and don’t really provide an effective solution on an individual handset. As a user then many of the actions described here can provide comparable functionality and protection. In the absence of a holistic solution then the enterprise or user must create a comprehensive suite of security controls and applications. The challenge here is maintaining that balance whereby security is seen as an enabler and does not impact too significantly everyday use of the device – failure to do so will lead to circumnavigation of security controls. As part of security education and awareness it would be advisable to discuss some of the core security implications associated with a Smartphone. Providing a suite of tools which can be installed on to a device, or offering an encrypted preloaded SD card, will ensure that exponential growth in mobile malware does not affect user’s organization. A company is only as secure as the weakest supplier or user, and mobile devices create all kinds of opportunities for malicious activity – for cybercriminals the path of least resistance is going to be the most tempting, and in such a new technology area there are plenty of potential exploits and attack vectors, both known and unknown, to take advantage of.

# References

1. [Android Documentation](https://developer.android.com/docs)
2. [Android Operating System: Architecture, Security Challenges and Solutions](https://www.researchgate.net/publication/299394606_Android_Operating_System_Architecture_Security_Challenges_and_Solutions)
3. [Google I/O 2008 - Dalvik Virtual Machine Internals](https://www.youtube.com/watch?v=ptjedOZEXPM&t=2s)
4. [Android OS - Processes and the Zygote!](http://coltf.blogspot.com/p/android-os-processes-and-zygote.html)