
UNIT 3 CASE STUDY - ANDROID

Structure

- 3.0 Introduction
- 3.1 Objectives
- 3.2 Salient Features of Android
- 3.3 Evolution of Android
- 3.4. Layered Architecture of Android OS
- 3.5. Processes and Threads Management in Android
- 3.6 Memory Management in Android
- 3.7 File Management in Android
- 3.8 Security Features in Android
- 3.9 Summary
- 3.10 Solutions/Answers
- 3.11 Further Readings

3.0 INTRODUCTION

Across the globe, in more than 190 countries, hundreds of millions of mobile devices are powered by Android operating system. It has strong user base and conquered around 75% of the global market share by the end of 2020. Google sponsored the project at initial stages and in the year 2005, it acquired the whole company. In September 2008, the first Android-powered device was launched in the market. It's popular because it has long list of features, user-friendly, has huge community support, provides a greater extent of customization, and a large number of companies build Android-compatible smart phones. At first, the purpose of Android was thought of as a mobile operating system. However, with the advancement of code libraries and its popularity among developers of the divergent domain, Android becomes an absolute set of software for all devices like tablets, wearables, set-top boxes, smart TVs, notebooks, etc. Android is an open-source operating system based on modified version of Linux kernel with a Java programming interface for mobile devices such as Smartphone (Touch Screen Devices who supports Android OS) as well for Tablets too.

This unit provides a case study on Android Operating System. It provides basic structure, features, process, file and memory management techniques it follows.

3.1 OBJECTIVES

After completing this unit, you will be able to:

- Understand the various features of the Android system
- Know the evolution of the Android OS

- Describe the architecture of the ANDROID OS.
- Know the Process and Memory management approaches
- Understand the File management in Android
- Identify the security features in Android

3.2 SALIENT FEATURES OF ANDROID OS

Android is a powerful open-source operating system that provides enormous features and some of them are listed below:

- **User Interface:** Android provides a great user interface which allow user to communicate with his device. It has user friendly UI which allow the user ease of access and can learn and operate the device quickly.
- **Storage:** Uses “SQLite” to store the data. SQLite being a relational data base processing queries are fast.
- **File Manager:** Data storage related activities will be managed by the file manager.
- **Media:** Supports wide range of media like JPEF, PNG, GIF, BMP, Ogg, MIDI, MP3, ACC 5.1, Vorbis, WAV, AMR, AMR-WB etc.
- **Messaging:** Provides various messaging services like MMS, SMS.
- **Connectivity:** Android provide various range of connectivity like GSM.EDGE, CDMA, Wi-Fi, LTE, NFC WiMAX, UMTS, EV-DO, IDEN
- **GPS:** It contains multiple APIs to support location-tracking services such as GPS.
- **Multitasking:** Android provides multitasking capability for the user. They can operate multiple applications at the same time.
- **Multi touch:** Android support multi touch functionality where user is allowed to make multiple touches at the same time and the OS will process it this is majorly used for gaming.
- **Widgets:** Android OS provides widgets which allow the user to have the organize the apps based on their requirements. It also allows the user to add, remove, resize, expand and collapse based on users usage and their need.
- **Screen Capture:** Allows the user to capture the screen shot of their mobile screen.
- **Near Field Communication:** Android provides a short range of wireless connection, which allows various devices to communicate quickly within the range. It is majorly used in payment systems.
- **Recording Capability:** It supports multimedia hardware control to perform playback or recording using a camera and microphone.
- **Webkit Layout:** Android has an integrated open-source WebKit layout based web browser to support User Interface like HTML5, CSS3.

- **Virtual Reality:** It provides support for virtual reality or 2D/3D Graphics.
- **Customization:** Customization is possible in Android OS based on our requirements.

3.3 EVOLUTION OF ANDROID OS

Andy Rubin founded Android Inc in October 2003 and later Google acquired Android Inc. Its initial intention is for camera as they have less market, it later changed to support mobile devices. Google announced the development of Android operating system in 2007 and Android launches its first Android mobile device. Android Inc. was acquired by Google in August, 2005. Key employees of Android Inc., including Andy Rubin, Rich Miner and Chris White, stayed at the company after the acquisition. After acquisition, a team led by Rubin developed a mobile device platform implemented using Linux kernel. Google marketed the platform to handset makers and carriers on the promise of providing a flexible, upgradable system.

Android operating systems has several versions. Following are the list of these versions depicted in the table 1:

Table 1: Android Versions

Code Name	Version	Release Date
Apple Pie	Android 1.0	September 23, 2008
Banana Bread	Android 1.1	February 9, 2009
Cupcake	Android 1.5	April 30, 2009
Donut	Android 1.6	September 15, 2009
Eclair	Android 2.0 – 2.1	October 26, 2009
Froyo	Android 2.2 – 2.2.3	May 20, 2010
Gingerbread	Android 2.3 – 2.3.4	December 6, 2010
Honeycomb	Android 3.0.x – 3.2.x	February 22, 2011
Ice Cream Sandwich	Android 4.0 – 4.0.4	October 18, 2011
Jelly Bean	Android 4.1 – 4.1.2	July 9, 2012
Kitkat	Android 4.4 – 4.4.4	July 9, 2012
Lollipop	Android 5.0 – 5.1	October 17, 2014
Marshmallow	Android 6.0 – 6.0.1	October 5, 2015
Nougat	Android 7.0 – 7.1	August 22, 2016
Oreo	Android 8.0	August 21, 2017
Pie	Android 9.0	August 6, 2018
Android Q	Android 10.0	September 3, 2019
Android 11	Android 11.0	September 8, 2020

In the next section let us study the Layered Architecture of Android.

3.4 LAYERED ARCHITECTURE OF ANDROID OS

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 given at Figure 1.

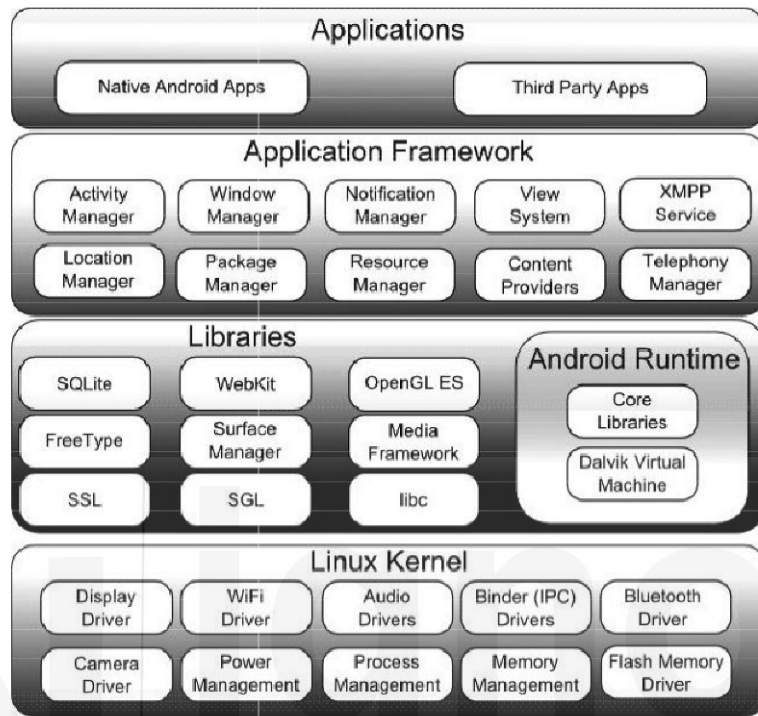


Figure 1: Architecture of Android OS (Source: developer.android.com)

3.4.1 Linux Kernel

Linux is the bottom layer. The bottom layer offers a level of abstraction between hardware and it contains all the essentials required for hardware drivers 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.

3.4.2 Libraries

A rich set of libraries are present on top of Linux kernel. Along with open-source Library Web browser engine WebKit, a collection of popular libraries like libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc. Java based libraries are used for the purpose of development of Android OS. The application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access.

A summary of some key core Android libraries available to the Android developer is as follows:

android.app	Provides access to the application model and is the cornerstone of all Android applications.
android.content	Facilitates content access, publishing and messaging between applications and application components.

android.database	Used to access data published by content providers and includes SQLite database management classes.
android.opengl	A Java interface to the OpenGL ES 3D graphics rendering API.
android.os	Provides applications with access to standard operating system services including messages, system services and inter-process communication
android.text	Used to render and manipulate text on a device display.
android.view	The fundamental building blocks of application user interfaces.
android.widget	A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
android.webkit	A set of classes intended to allow web-browsing capabilities to be built into applications.

3.4.3 Android Runtime

This section is the part of the Libraries layer which contains a key component called Dalvik Virtual Machine (Dalvik VM) which is an optimized Android library acts like a of Java Virtual Machine. The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The 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.

3.4.4 Application Framework

Several high level services will be provided to the applications through Application Framework layer. These application services are available in the form of Java classes. Application developers can use these services in their applications.

The Android framework includes the following key services:

- **Activity Manager** “ Controls all aspects of the application lifecycle and activity stack.
- **Content Providers** “ Allows applications to publish and share data with other applications.
- **Resource Manager** “ Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
- **Notifications Manager** “ Allows applications to display alerts and notifications to the user.
- **Windows Manager** – Allows applications to apply various operations on windows like create, minimize, resize etc.
- **View System** “ An extensible set of views used to create application user interfaces.

3.4.5 Applications

The top layer of Android Architecture is Applications-layer. Users have to install their own applications on this layer. Browser, Games, Contacts Books, etc., are some of the examples of such applications. The application layer runs within the Android run time using the classes and services made available from the application framework.

3.5 PROCESSES AND THREADS MANAGEMENT IN ANDROID

Android create process for each application, all the components are running from a main thread in low power and low memory. Android automatically manages the processes and kills the least used or inactive process.

3.5.1 Process Life Cycle

The Android system tries to maintain an application process for as long as possible, but eventually needs to remove old processes to reclaim memory for new or more important processes. To determine which processes to keep and which to kill, the system places each process into an “importance hierarchy” based on the components running in the process and the state of those components. Processes with the lowest importance are eliminated first, then those with the next lowest importance, and so on, as necessary to recover system resources. There are five levels in the importance hierarchy. The following list presents the different types of processes in order of importance:

(i) Foreground process

A process that is required for what the user is currently doing. A process is considered to be in the foreground if any of the following conditions are true:

- It hosts an Activity that the user is interacting with
- It hosts a Service that’s bound to the activity that the user is interacting with.
- It hosts a Service that’s running “in the foreground” and the service has called `startForeground()`.
- It hosts a Service that’s executing one of its lifecycle callbacks (`onCreate()`, `onStart()`, or `onDestroy()`).
- It hosts a `BroadcastReceiver` that’s executing its `onReceive()` method.

Generally, only a few foreground processes exist at any given time. They are killed only as a last resort, if memory is so low and cannot continue to run. Generally, at that point, the device has reached a memory paging state, so killing some foreground processes is required to keep the user interface responsive.

(ii) Visible process

A process that doesn’t have any foreground components, but still can affect what the user sees on screen. A process is considered to be visible if either of the following conditions are true:

- It hosts an Activity that is not in the foreground, but is still visible to the user (its `onPause()` method has been called).

- It hosts a Service that's bound to a visible (or foreground) activity.

A visible process is considered extremely important and will not be killed unless doing so is required to keep all foreground processes running.

(iii) Service process

A process that is running a service that has been started with the `startService()` method and does not fall into either of the two higher categories. Although service processes are not directly tied to anything the user sees, they are generally doing things that the user cares about such as playing music in the background or downloading data on the network, so the system keeps them running unless there's not enough memory to retain them along with all foreground and visible processes.

(iv) Background process

A process holding an activity that's not currently visible to the user (the activity's `onStop()` method has been called). These processes have no direct impact on the user experience, and the system can kill them at any time to reclaim memory for a foreground, visible, or service process. Usually there are many background processes running, so they are kept in an LRU (least recently used) list to ensure that the process with the activity that was most recently seen by the user is the last to be killed. If an activity implements its lifecycle methods correctly, and saves its current state, killing its process will not have a visible effect on the user experience, because when the user navigates back to the activity, the activity restores all of its visible state.

(v) Empty process

A process that doesn't hold any active application components is called empty process. The only reason to keep this kind of process alive is for caching purposes, to improve startup time the next time a component needs to run in it. The system often kills these processes in order to balance overall system resources between process caches and the underlying kernel caches.

Android ranks a process at the highest level it can, based upon the importance of the components currently active in the process.

3.5.2 Threads

When an application is launched, the system creates a thread of execution for the application, called "main". This thread is very important because it is in charge of dispatching events to the appropriate user interface widgets, including drawing events. It is also the thread in which your application interacts with components from the Android UI toolkit. This 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, and system calls to each component are dispatched from that thread. Consequently, methods that respond to system callbacks (such as `onKeyDown()` to report user actions or a lifecycle callback method) always run in the UI thread of the process.

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 model can yield poor performance unless you implement your application properly. 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. Even worse, if the UI thread is blocked for more than a few seconds (about 5 seconds currently) the user is presented with the infamous "application not responding" (ANR) dialog. The user might then decide to quit your application and uninstall it if they are unhappy. Additionally, the Android UI toolkit is not thread-safe.

Interprocess Communication

Android offers a mechanism for Interprocess Communication (IPC) using remote procedure calls (RPCs), in which a method is called by an activity or other application component, but executed remotely (in another process), with any result returned back to the caller. This entails decomposing a method call and its data to a level the operating system can understand, transmitting it from the local process and address space to the remote process and address space, then reassembling and reenacting the call there. Return values are then transmitted in the opposite direction. Android provides all the code to perform these IPC transactions, so you can focus on defining and implementing the RPC programming interface. To perform IPC, your application must bind to a service, using `bindService()`.

In the next section we will study about the Memory Management in Android OS.

3.6 MEMORY MANAGEMENT IN ANDROID

The Android Runtime (ART) and Dalvik Virtual Machine (DVM) use paging and memory-mapping (*mmapping*) to manage memory. This means that any memory an app modifies; whether by allocating new objects or touching *mmapped* pages, they remains resident in RAM and cannot be paged out. The only way to release memory from an app is to release object references that the app holds, making the memory available to the garbage collector. That is with one exception: any files *mmapped* in without modification, such as code, can be paged out of RAM if the system wants to use that memory elsewhere.

3.6.1 Shared Memory

In order to fit everything it needs in RAM, Android tries to share RAM pages across processes. It can do so in the following ways:

- Each app process is forked from an existing process called Zygote. The Zygote process starts when the system boots and loads common framework code and resources (such as activity themes). To start a new app process, the system forks the Zygote process then loads and runs the app's code in the new process. This approach allows most of the RAM pages allocated for framework code and resources to be shared across all app processes.
- Most static data is *mmapped* into a process. This technique allows data to be shared between processes, and also allows it to be paged out when needed. Example static data include: Dalvik code (by placing it in a pre-

linked `.odex` file for direct *mmapping*), app resources (by designing the resource table to be a structure that can be *mmapped* and by aligning the zip entries of the APK), and traditional project elements like native code in `.so` files.

- In many places, Android shares the same dynamic RAM across processes using explicitly allocated shared memory regions. For example, window surfaces use shared memory between the app and screen compositor, and cursor buffers use shared memory between the content provider and client.

3.6.2 Types of Memory

Android devices contain three different types of memory: RAM, zRAM, and storage. Note that both the CPU and GPU access the same RAM.

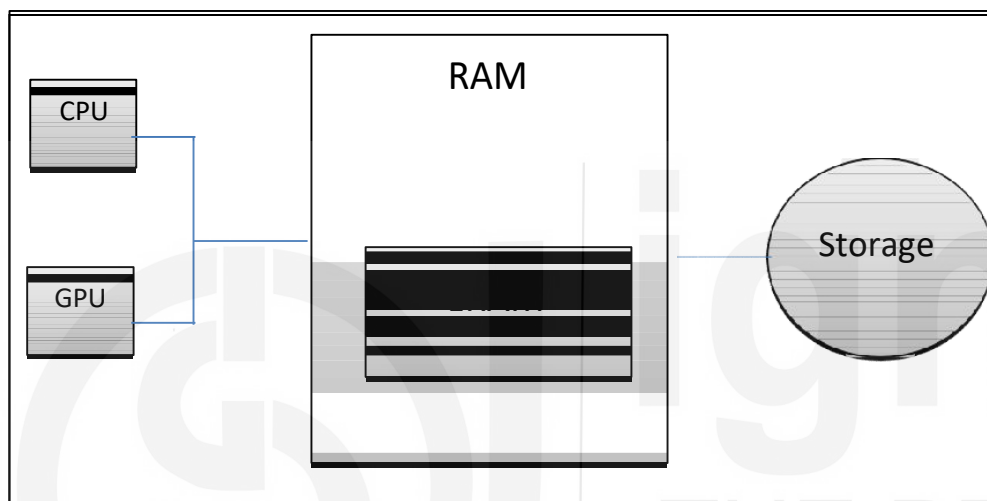


Figure 2: Types of Memory - RAM, zRAM, and Storage

- RAM is the fastest type of memory, but is usually limited in size. High-end devices typically have the largest amounts of RAM.
- zRAM is a partition of RAM used for swap space. Everything is compressed when placed into zRAM, and then decompressed when copied out of zRAM. This portion of RAM grows or shrinks in size as pages are moved into or taken out of zRAM. Device manufacturers can set the maximum size.
- Storage contains all of the persistent data such as the file system and the included object code for all apps, libraries, and the platform. Storage has much more capacity than the other two types of memory. On Android, storage isn't used for swap space like it is on other Linux implementations since frequent writing can cause wear on this memory, and shorten the life of the storage medium.

3.6.3 Memory Allocation Among Processes

The Android platform runs on the premise that free memory is wasted memory. It tries to use all of the available memory at all times. For example, the system keeps apps in memory after they've been closed so the user can quickly switch back to them. For this reason, Android devices often run with very little free memory. Memory management is vital to properly allocate memory among important system processes and many user applications.

3.6.4 Paging

RAM is broken up into *pages*. Typically each page is 4KB of memory. Pages are considered either *free* or *used*. Free pages are unused RAM. Used pages are RAM that the system is actively using, and are grouped into the following categories:

- **Cached:** Memory backed by a file on storage (for example, code or memory-mapped files). There are two types of cached memory:
- **Private:** Owned by one process and not shared.
- **Clean:** Unmodified copy of a file on storage, can be deleted by *kswapd* to increase free memory.
- **Dirty:** Modified copy of the file on storage; can be moved to, or compressed in, zRAM by *kswapd* to increase free memory or allows changes to be written back to the file in storage to increase free memory by *kswapd*, or explicitly using `msync()` or `munmap()` or Can be moved/compressed in zRAM by *kswapd* to increase free memory
- **Shared:** Used by multiple processes.
- **Anonymous:** Memory **not** backed by a file on storage (for example, allocated by `mmap()` with the `MAP_ANONYMOUS` flag set).

3.6.5 Low Memory Management

Android has two main mechanisms to deal with low memory situations:

- The kernel swap daemon
- Low-memory killer

(i) kernel swap daemon

The kernel swap daemon (*kswapd*) is part of the Linux kernel, and converts used memory into free memory. The daemon becomes active when free memory on the device runs low. The Linux kernel maintains low and high free memory thresholds. When free memory falls below the low threshold, *kswapd* starts to reclaim memory. Once the free memory reaches the high threshold, *kswapd* stops reclaiming memory. *kswapd* can reclaim clean pages by deleting them because they're backed by storage and have not been modified. If a process tries to address a clean page that has been deleted, the system copies the page from storage to RAM. This operation is known as *demand paging*.

kswapd can move cached private dirty pages and anonymous dirty pages to zRAM, where they are compressed. Doing so frees up available memory in RAM (free pages).

If a process tries to touch a dirty page in zRAM, the page is uncompressed and moved back into RAM. If the process associated with a compressed page is killed, then the page is deleted from zRAM. If the amount of free memory falls below a certain threshold, the system starts killing processes.

(ii) Low-Memory Killer

Many times, kswapd cannot free enough memory for the system. In this case, the system uses `onTrimMemory()` to notify an app that memory is running low and that it should reduce its allocations. If this is not sufficient, the kernel starts killing processes to free up memory. It uses the low-memory killer (LMK) to do this.

To decide which process to kill, LMK uses an “out of memory” score called `oom_adj_score` to prioritize the running processes. Processes with a high score are killed first. Background apps are first to be killed, and system processes are last to be killed. The following table lists the LMK scoring categories from high-to-low. Items in the highest-scoring category, in row one, will be killed first:

These are descriptions for the various categories in the table above:

- **Background Apps:** Apps that were run previously and are not currently active. LMK will first kill background apps starting with the one with the highest `oom_adj_score`.
- **Previous App:** The most recently-used background app. The previous app has higher priority (a lower score) than the background apps because it is more likely the user will switch to it than one of the background apps.
- **Home App:** This is the launcher app. Killing this will make the wallpaper disappear.
- **Services:** Services are started by applications and may include syncing or uploading to the cloud.
- **Perceptible Apps:** Non-foreground apps that are perceptible to the user in some way, such as running a search process that displays a small UI or listening to music.
- **Foreground App:** The app currently being used. Killing the foreground app looks like an application crash which might indicate to the user that something is going wrong with the device.
- **Persistent (services):** These are core services for the device, such as telephony and wifi.
- **System:** System processes. As these processes are killed, the phone may appear to reboot.
- **Native:** Very low-level processes used by the system (for example, kswapd).

3.6.6 Garbage Collection

A managed memory environment, like the ART or DVM, keeps track of each memory allocation. Once it determines that a piece of memory is no longer being used by the program, it frees it back to the heap, without any intervention from the programmer. The mechanism for reclaiming unused memory within a managed memory environment

is known as *garbage collection*. Garbage collection has two goals: find data objects in a program that cannot be accessed in the future; and reclaim the resources used by those objects.

Android's memory heap is a generational one, meaning that there are different buckets of allocations that it tracks, based on the expected life and size of an object being allocated. For example, recently allocated objects belong in the *Young generation*. When an object stays active long enough, it can be promoted to an older generation, followed by a permanent generation.

3.7 FILE MANAGEMENT IN ANDROID

Android uses different partitions to handle files and folders on the system like windows operating system. Each of these partitions has its own functionality. Mainly, there are six specific partitions in the file system of Android devices. Each model may have their own organization for this file system partitioning. But most of the Android Devices have the following list of partitions logically in common like: */boot*, */system*, */recovery*, */data*, */cache* and */misc*. The following are two separate partitions for the SD card of Android Devices */sdcard* and */sd-ext*.

/boot

- This is the boot partition of your Android device, as the name suggests.
- It includes the android kernel and the ramdisk.
- The device will not boot without this partition.
- Wiping this partition from recovery should only be done if absolutely required and once done, the device must NOT be rebooted before installing a new one, which can be done by installing a ROM that includes a */boot* partition.

/system

- As the name suggests, this partition contains the entire Android OS.
- This includes the Android GUI and all the system applications that come pre-installed on the device.
- Wiping this partition will remove Android from the device without rendering it unbootable, and we can still be able to place the phone into recovery or bootloader mode to install a new ROM.

/recovery

- This is specially designed for backup.
- The recovery partition can be considered as an alternative boot partition, that lets the device boot into a recovery console for performing advanced recovery and maintenance operations on it.

/data

- It is called user data partition.
- This partition contains the user's data like your contacts, sms, settings and all android applications that you have installed.

- While we are doing factory reset on our device, this partition will wipe out, then the device will be in the state, when we use for the first time, or the way it was after the last official or custom ROM installation.

/cache

- This is the partition where Android stores frequently accessed data and app components.
- Wiping the cache doesn't affect our personal data but simply gets rid of the existing data there, which gets automatically rebuilt as we continue using the device.

/misc

- This partition contains miscellaneous system settings in form of on/off switches.
- These settings may include CID (Carrier or Region ID), USB configuration and certain hardware settings etc.
- This is an important partition and if it is corrupt or missing, several of the device's features will not function normally.

/sdcard

- This is not a partition on the internal memory of the device but rather the SD card.
- In terms of usage, this is our storage space to use as you see fit, to store media, documents, ROMs etc. on it.
- Wiping it is perfectly safe as long as we backup all the data you require from it, to the computer first.
- Though several user-installed apps save their data and settings on the SD card and wiping this partition will make we lose all that data.

/sd-ext

- This is not a standard Android partition, but has become popular in the custom ROM scene.
- It is basically an additional partition on your SD card that acts as the /data partition.
- It is especially useful on devices with little internal memory allotted to the /data partition.
- Thus, users who want to install more programs than the internal memory allows can make this partition and use it for installing their apps.

3.8 SECURITY FEATURES IN ANDROID

Android incorporates industry-leading security features and works with developers and device implementers to keep the Android platform and ecosystem safe. A robust security model is essential to enable a vigorous ecosystem of apps and devices built on and around the Android platform and supported by cloud services. As a result, through

its entire development lifecycle, Android has been subject to a rigorous security program. The key components of the Android Security include:

3.8.1 System and kernel security

At the operating system level, the Android platform provides the security of the Linux kernel, as well as a secure inter-process communication (IPC) facility to enable secure communication between applications running in different processes. These security features at the OS level ensure that even native code is constrained by the Application Sandbox. Whether that code is the result of included application behavior or an exploitation of application vulnerability, the system is designed to prevent the rogue application from harming other applications, the Android system, or the device itself.

The foundation of the Android platform is the Linux kernel. Linux has become a stable and secure kernel trusted by many corporations and security professionals. As the base for a mobile computing environment, the Linux kernel provides Android with several key security features, including:

- A user-based permissions model
- Process isolation
- Extensible mechanism for secure IPC
- The ability to remove unnecessary and potentially insecure parts of the kernel

As a multiuser operating system, a fundamental security objective of the Linux kernel is to isolate user resources from one another. The Linux security philosophy is to protect user resources from one another.

3.8.2 Verified boot

Android 6.0 and later supports verified boot and device-mapper-verity. Verified boot guarantees the integrity of the device software starting from a hardware root of trust up to the system partition. During boot, each stage cryptographically verifies the integrity and authenticity of the next stage before executing it. Android 7.0 and later supports strictly enforced verified boot, which means compromised devices cannot boot.

3.8.3 File System Permissions

In a UNIX-style environment, file-system permissions ensure that one user cannot alter or read another user's files. In the case of Android, each application runs as its own user. Unless the developer explicitly shares files with other applications, files created by one application cannot be read or altered by another application.

3.8.4 Platform Security

Android seeks to be the most secure and usable operating system for mobile platforms by repurposing traditional operating system security controls to Protect app and user data, Protect system resources (including the network), Provide app isolation from the system, other apps, and from the user. To achieve these objectives, Android provides these key security features:

- Robust security at the OS level through the Linux kernel
- Mandatory app sandbox for all apps

- Secure interprocess communication
- App signing
- App-defined and user-granted permissions

3.8.5 Program Level Security Features

- **Design review:** The Android security process begins early in the development lifecycle with the creation of a rich and configurable security model and design. Each major feature of the platform is reviewed by engineering and security resources, with appropriate security controls integrated into the architecture of the system.
- **Penetration testing and code review:** During the development of the platform, Android-created and open source components are subject to vigorous security reviews. These reviews are performed by the Android Security Team, Google's Information Security Engineering team, and independent security consultants. The goal of these reviews is to identify weaknesses and possible vulnerabilities well before major releases, and to simulate the types of analysis that are performed by external security experts upon release.
- **Open source and community review:** AOSP enables broad security review by any interested party. Android also uses open source technologies that have undergone significant external security review, such as the Linux kernel. Google Play provides a forum for users and companies to provide information about specific apps directly to users.
- **Incident response:** Even with these precautions, security issues may occur after shipping, which is why the Android project has created a comprehensive security response process. Full-time Android security team members monitor the Android-specific and the general security community for discussion of potential vulnerabilities and review security bugs filed on the Android bug database. Upon the discovery of legitimate issues, the Android team has a response process that enables the rapid mitigation of vulnerabilities to ensure that potential risk to all Android users is minimized. These cloud-supported responses can include updating the Android platform (AOSP updates), removing apps from Google Play, and removing apps from devices in the field.
- **Monthly security updates:** The Android security team provides monthly updates to Google Android devices and all our device manufacturing partners.

3.8.6 Cryptography

Android provides a set of cryptographic APIs for use by applications. These include implementations of standard and commonly used cryptographic primitives such as AES, RSA, DSA, and SHA. Additionally, APIs are provided for higher level protocols such as SSL and HTTPS.

3.8.7 User Security Features

This can be categorized into File System Encryption and Password Protection.

(i) File System Encryption

Android 3.0 and later provides full file-system encryption, so all user data can be

encrypted in the kernel. Android 5.0 and later supports full disk encryption. Full-disk encryption uses a single key—protected with the user’s device password—to protect the whole of a device’s user data partition. Upon boot, users must provide their credentials before any part of the disk is accessible.

Android 7.0 and later supports file-based encryption. File-based encryption allows different files to be encrypted with different keys that can be unlocked independently.

(ii) Password Protection

Android can be configured to verify a user-supplied password prior to providing access to a device. In addition to preventing unauthorized use of the device, this password protects the cryptographic key for full file system encryption. Use of a password and/or password complexity rules can be required by a device administrator.

3.8.8 Device Security

Android 2.2 and later provide the Android Device Administration API, which provides device administration features at the system level. For example, the built-in Android Email application uses the APIs to improve Exchange support. Through the Email application, Exchange administrators can enforce password policies including alphanumeric passwords or numeric PINs across devices. Administrators can also remotely wipe (that is, restore factory defaults on) lost or stolen handsets.

☞ Check Your Progress 1

- 1) Mention and explain briefly the main building blocks of Android platform.

.....
.....
.....
.....

- 2) Describe the new IPC mechanisms provided by Android.

.....
.....
.....
.....

3.9 SUMMARY

Android is a powerful open-source operating system which provides a lot of great features. Its open-source and we can customize the OS based on our requirements. It supports connectivity for GSM, CDMA, WIFI, NFC, Bluetooth, etc. for telephony or data transfer. It will allow us to make or receive a calls / SMS messages and we can send or retrieve data across mobile networks. By using WIFI technology we can pair with other devices using apps. Android has multiple APIs to support location-based services such as GPS. We can perform all data storage related activities by using lightweight database SQLite. It has a wide range of media supports like AVI, MKV,

FLV, MPEG4, etc. to play or record a variety of audio/video and having a different image format like JPEG, PNG, GIF, BMP, MP3, etc. . It has extensive support for multimedia hardware control to perform playback or recording using camera and microphone. It has an integrated open-source WebKit layout based web browser to support HTML5, CSS3. It supports a multi-tasking, we can move from one task window to another and multiple applications can run simultaneously. It will give a chance to reuse the application components and the replacement of native applications. We can access the hardware components like Camera, GPS, and Accelerometer. It has support for 2D/3D Graphics.

In this unit, we have discussed features of Android OS, Architecture, process management, memory management, file system and security aspects in Android operating system.

3.10 SOLUTIONS/ANSWERS

Check Your Progress 1:

1) The main Android platform building blocks are:

- **Device hardware:** Android runs on a wide range of hardware configurations including mobile phones, tablets, watches, automobiles, smart TVs, OTT gaming boxes, and set-top-boxes. Android is processor-agnostic, but it takes advantage of some hardware-specific security capabilities such as ARM eXecute-Never.
- **Android operating system:** The core operating system is built on top of the Linux kernel. All device resources, like camera functions, GPS data, Bluetooth functions, telephony functions, and network connections are accessed through the operating system.
- **Android Application Runtime:** Android apps are most often written in the Java programming language and run in the Android runtime (ART). However, many apps, including core Android services and apps, are native apps or include native libraries. Both ART and native apps run within the same security environment, contained within the Application Sandbox. Apps get a dedicated part of the file system in which they can write private data, including databases and raw files.

Android apps extend the core Android operating system. There are two primary sources for apps:

- **Preinstalled apps:** Android includes a set of preinstalled apps including phone, email, calendar, web browser, and contacts. These function as user apps and they provide key device capabilities that can be accessed by other apps. Preinstalled apps may be part of the open source Android platform, or they may be developed by a device manufacturer for a specific device.
- **User-installed apps:** Android provides an open development environment that supports any third-party app. Google Play offers users hundreds of thousands of apps.

Processes can communicate using any of the traditional UNIX-type mechanisms. Examples include the file-system, local sockets, or signals. However, the Linux permissions still apply.

- 2) Android also provides new IPC mechanisms:
- **Binder:** A lightweight capability-based remote procedure call mechanism designed for high performance when performing in-process and cross-process calls. Binder is implemented using a custom Linux driver.
 - **Services:** Services (discussed above) can provide interfaces directly accessible using binder.
 - **Intents:** An Intent is a simple message object that represents an “intention” to do something. For example, if your application wants to display a web page, it expresses its “Intent” to view the URL by creating an Intent instance and handing it off to the system. The system locates some other piece of code (in this case, the Browser) that knows how to handle that Intent, and runs it. Intents can also be used to broadcast interesting events (such as a notification) system-wide.
 - **ContentProviders:** A ContentProvider is a data storehouse that provides access to data on the device; the classic example is the ContentProvider that is used to access the user’s list of contacts. An application can access data that other applications have exposed via a ContentProvider, and an application can also define its own ContentProviders to expose data of its own.

While it is possible to implement IPC using other mechanisms such as network sockets or world-writable files, these are the recommended Android IPC frameworks. Android developers will be encouraged to use best practices around securing users’ data and avoiding the introduction of security vulnerabilities.

3.11 FURTHER READINGS

1. Barry Burd, Android Application Development All-in-One for Dummies, Second Edition, Wiley, 2015.
2. Pradeep Kothari, Android Application Development, Black Book, Kindle Edition(Dreamtech Press), 2019.
3. Pratiyush Guleria, Android for Beginners: Learn Step-by-Step, BPB Publications, 2018.
4. developer.android.com