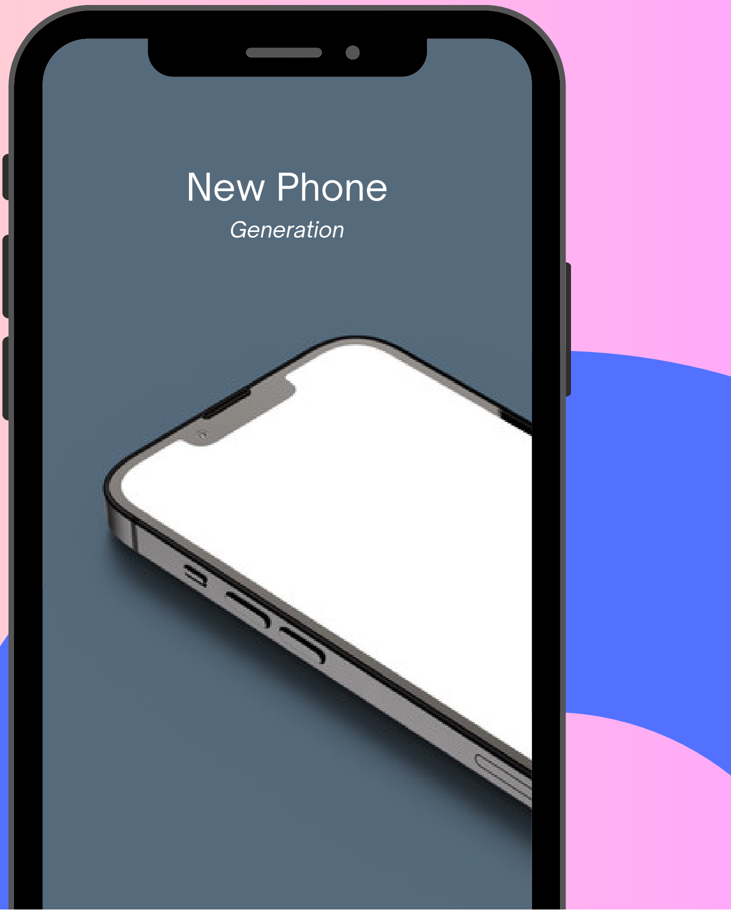
android

OS CASE STUDY ANDROID OS

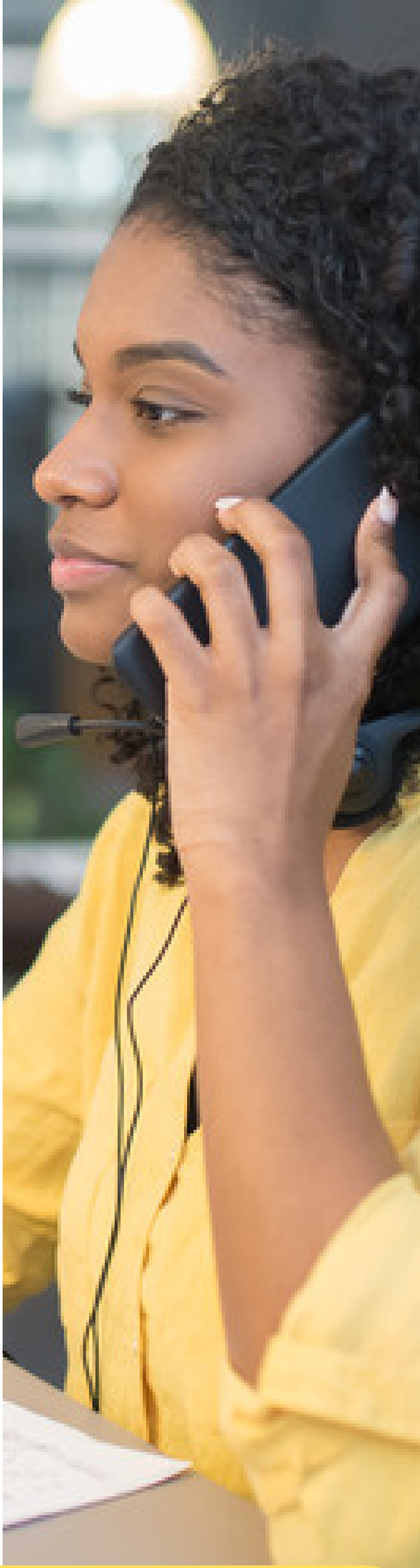
Submitted By:

Vishu Aasliya
2021UCA1929

Saumya Karnwal
2021UCA1879



INDEX



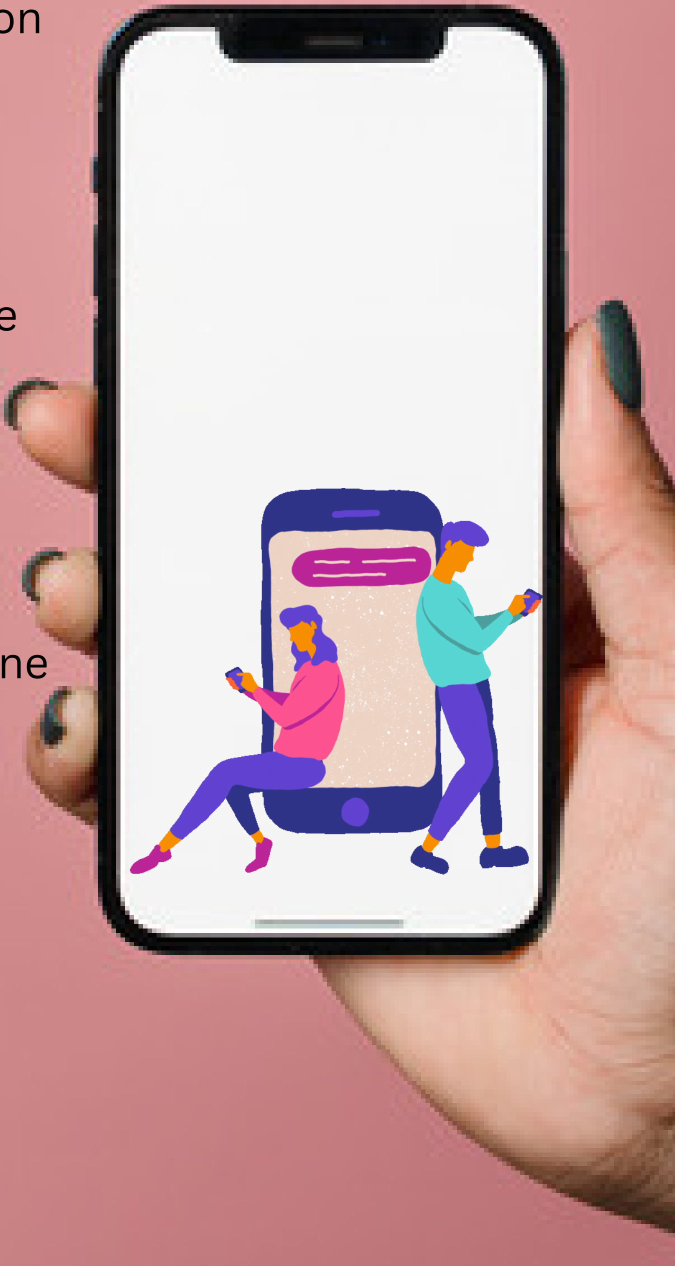
Introduction

Android OS is an **open-source operating system (OS)** for mobile devices like tablets and smartphones. It was created by a group of hardware, software, and telecommunications companies called the **Open Handset Alliance** and **Google**. Android uses a **modified form of the Java programming language** and is **based on the Linux kernel**.

The ability of Android to **customize the user interface (UI)** is one of its distinctive features. **Widgets**, which are compact programs, can be added to the home screen to offer quick access to frequently used features, make this possible. The enormous selection of apps that are available for download from the Google Play Store is another important aspect of Android. Android provides users with a wide range of options for everything from productivity and entertainment to communication, gaming, with over 3 million apps readily available.

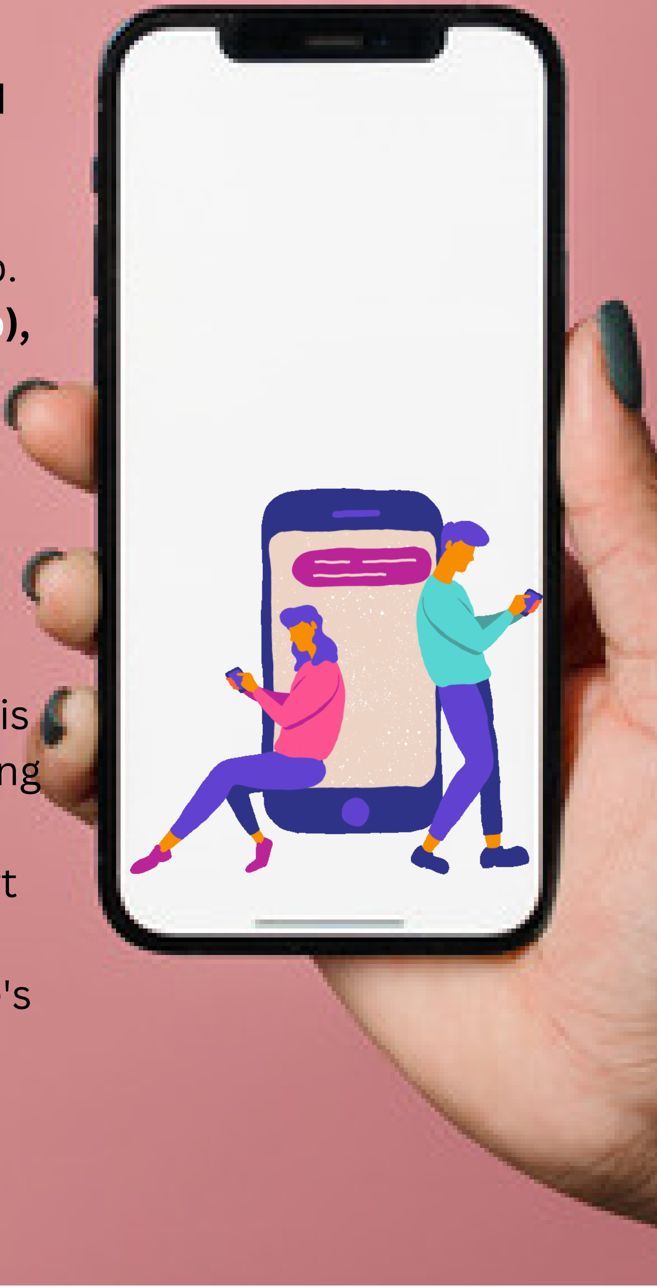
A **variety of hardware**, such as cameras, sensors, and wireless standards like Bluetooth and Wi-Fi, are **supported by Android natively**. Programmers can create apps that make use of these features to give users a richer experience.

Android is **used in a variety of devices besides smartphones and tablets, including smart TVs, smartwatches, and even automobiles**. Android has become one of the most widely used mobile operating systems in the world thanks to its **adaptability, customization options, and strong app ecosystem.**



History

- The beginning of Android OS can be traced to **2003**, when a team of programmers **under the direction of Andy Rubin** established a business known as **Android Inc**. Their initial objective was to create an operating system for digital cameras, but as soon as they realized there wasn't much of a market for cameras, they switched their attention to mobile devices.
- After **Google acquired Android Inc. in 2005**, the creation of the Android OS got under way seriously. Android 1.0, also known as **Android 1.0, was introduced in September 2008**. It had functions like a web browser, Google Maps, and a camera app and was primarily made for smartphones.
- A number of new features were added to **Android 1.5, also known as Cupcake**, when it was released in **April 2009**. These included an on-screen keyboard, support for third-party widgets, and the capacity to record and view videos.



- In **October 2009, Google released Android 2.0 (Eclair)**, which added support for multiple accounts, enhanced messaging capabilities, and enhanced the camera app.
- **Android 4.4 (KitKat), Android 5.0 (Lollipop), Android 6.0 (Marshmallow), Android 7.0 (Nougat), Android 8.0 (Oreo), Android 9.0 (Pie),** and **Android 10** were subsequent releases that furthered the OS's development by introducing new features and enhancing performance.
- With over 2 billion active devices, Android is currently the most popular mobile operating system worldwide. The use of it in smartphones, tablets, smartwatches, smart TVs, and other devices has made it a necessary component of billions of people's daily lives worldwide.

Architecture

The Android operating system's architecture is mostly broken down into six smaller components.

- The Linux Kernel
- Hardware Abstraction Layer(HAL)
- Android Runtime
- Native C/C++ Libraries
- Java API Framework
- System Apps

The Linux Kernel: The Android platform is built on top of the Linux Kernel. A kernel is essentially a program that runs always in the system. The architecture of an android starts with the Linux kernel. Android 4.0 is now powered by Linux 4.4. Memory management, security management, process management, and device management are all supported by the Linux kernel. In order to interface with other peripheral devices, it also includes a list of device drivers. A device driver is a piece of software that connects to hardware devices via a software interface. For underpinning features like low-level memory management and multi-threading, the Android runtime depends on the Linux kernel.

Hardware Abstraction Layer: In essence, the Hardware abstraction layer acts as a link between hardware and software. It offers standardized interfaces that reveal hardware capabilities of the device to the more advanced Java API framework. It is a programming layer that enables a computer operating system to communicate with hardware at a more general or abstract level than at a particular hardware level. A device driver or the OS kernel both have the ability to call HAL. In either instance, the calling programme has a wider range of interactions with the device than it would otherwise have.

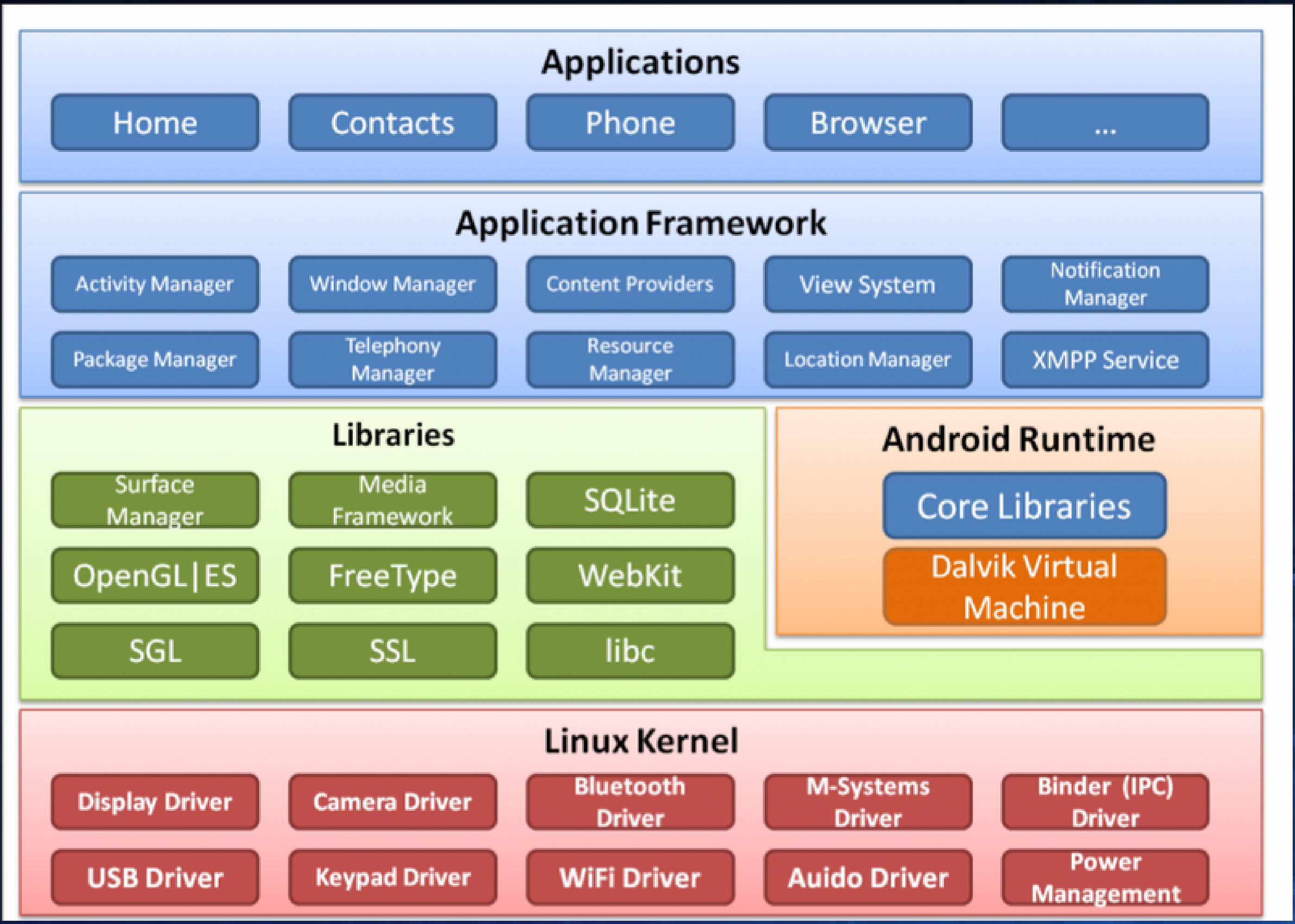
Android Runtime: The Dalvik virtual machine and a selection of the essential Android libraries make up the Android runtime. Android Runtime (ART), which is better optimized, is used in versions following Kit Kat. The key features of the Android runtime are enhanced debugging support and garbage collection, which are utilized to operate numerous virtual machines on low-memory devices. Because the Dalvik virtual machine uses the just-in-time (JIT) compiler technology, which compiles code to native code each time the application is run, the ART is more optimized than the DVM. In contrast, the ART uses the ahead-of-time (AOT) technology, which compiles code to native code only once during the installation process. Since the code is only compiled once, ART is superior to DVM. The benefits of ART include faster app startup times, better garbage collection, and smaller memory footprints, or less memory referenced by an application.

Native C/C++ Libraries: Because the Dalvik virtual machine uses the just-in-time (JIT) compiler technology, which compiles code to native code each time the application is run, the ART is more optimised than the DVM. In contrast, the ART uses the ahead-of-time (AOT) technology, which compiles code to native code only once during the installation process. Since the code is only compiled once, ART is superior to DVM. The benefits of ART include faster app startup times, better garbage collection, and smaller memory footprints, or less memory referenced by an application.

Java API Framework: With the aid of the Java API, which serves as a building block for developing android apps, all capabilities of the android OS are accessible to users and developers. The Java API framework offers a variety of elements and services, such as resource managers, notification managers, activity managers, and content providers, among others, that are beneficial when creating complex and modular applications.

System Apps: This section contains all of the programs that are available on Android devices, whether the user voluntarily install them or the device comes with the Android OS preloaded. These applications range in functionality from a basic calculator to a sophisticated map application.

Diagrammatic Representation of Android Architecture



Source: https://bhardwajmanish.com/wp-content/uploads/2020/06/android-framework1.png

Process Management

Any operating system must have effective process management, and Android is no different. To manage processes efficiently, guarantee optimal performance, and make the best use of available resources, Android OS employs a number of techniques.

The **process management system built into the Linux kernel** is one of the primary tools Android uses to manage processes. **Memory, CPU time, and I/O bandwidth are among the resources that the kernel is in charge of allocating to various processes.** Android expands on this framework by adding a number of new tools for efficient process management.

<u>Activity Manager:</u> The Activity Manager, which controls the lifecycle of activities and services inside an app, is one such mechanism. Services are background processes that operate independently of the user interface, whereas activities are the visible elements of an app. Depending on the user's actions and the resources available, the Activity Manager **makes sure that activities and services are started and stopped in a timely manner**.

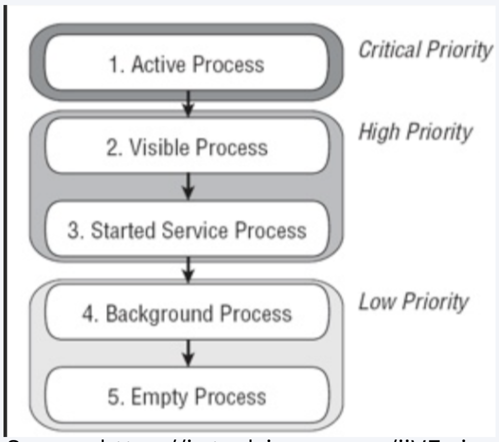
<u>The Low Memory Killer (LMK):</u> is yet another essential process management tool used by Android. When memory becomes limited, the LMK, a background process that **monitors the amount of free memory available, terminates low-priority processes**. As a result, the system is less likely to crash or become unresponsive due to memory exhaustion.

<u>Priority-based process scheduling system:</u> The order in which processes are executed on the CPU is also determined by a priority-based process scheduling system that Android uses. Various processes are given different priorities by the system based on their significance and resource needs. For instance, foreground processes, like the one that is currently running, are prioritized over background processes, which may be paused or terminated to free up resources.

<u>Process state freezing:</u> Additionally, "process state freezing" is a method used by Android to **boost efficiency and reduce power consumption**. In order to achieve this, background processes, like apps, are frozen in place and restored when they become active again. This speeds up app launch times and lessens the drain on the battery caused by background processes.
<u>Strong security model:</u> Finally, a strong security model is implemented in Android to **guarantee that processes are separated from one another and cannot access one another's data or resources without authorization**. Only the permissions necessary for each app to carry out its intended functions are granted, and each app runs in its own process. This makes it harder for malicious apps to access private information or interfere with other apps.

In conclusion, the platform uses a variety of mechanisms to manage processes effectively and guarantee optimal performance and resource utilization. Process management is a crucial aspect of Android OS. Android offers a stable and secure platform for users and developers alike by implementing a variety of techniques like process state freezing, priority-based scheduling, and security isolation.

Image:
Android Process Scheduling



Source: https://i.stack.imgur.com/iiVEa.jpg

Process Segregation: A key component of the Android OS process management system is **process isolation**. By doing this, it makes sure that every process has its own distinct environment, memory space, and system resources. This avoids process conflict, which can result in system failure and instability. For the purposes of identification and tracking, each process is also given a special process ID (PID).

Memory administration: Another crucial element of the Android OS process management system is memory management. The OS can dynamically allocate memory to processes as needed thanks to a method known as **virtual memory**. This prevents resource waste and boosts system performance by ensuring that processes only use the memory they require.

Priority Planning: Another essential component of the Android OS process management system is priority scheduling. Processes are given a priority according to their level of importance and the resources they need. This enables the OS to give priority to crucial operations like system services, ensuring that they receive the resources required for proper operation.

The process management system for the Android operating system is made up of a number of essential parts that cooperate to guarantee effective performance. These elements consist of:

- **Zygote** – The process management system of the Android OS relies heavily on the zygote. Initializing new processes is the responsibility of this background process. By preloading crucial system libraries and resources, it accomplishes this and speeds up the startup of new processes.
- **Activity Manager** - The Activity Manager is in charge of overseeing all of the running activities on the device. The active activity is always in the foreground thanks to the stack of activities it maintains. Additionally, it controls how activities are started, paused, resumed, and stopped throughout their lifecycle.
- **Package Manager** - The package manager is in charge of overseeing all of the packages and programs that have been installed on the device. It monitors installed packages and the processes that are connected to them to make sure they are properly initialized and managed.
- **Resource Manager** - The resource manager is in charge of overseeing the different system resources that processes use. The allocation of resources is optimized for effective performance, ensuring that processes only use what they require.

Conclusion: In conclusion, process management is an important component of every operating system, including the Android OS. Processes are isolated, memory usage is optimized, and priority scheduling is used for effective performance thanks to the system's well-designed process management. The different parts of the process management system cooperate to make sure the machine functions smoothly and effectively. Android OS has grown to be one of the most widely used mobile operating systems worldwide thanks to its powerful process management system.

Interprocess Communication

Interprocess communication (IPC) is a critical aspect of any operating system, such as Android. In Android, IPC is used to **facilitate communication between different processes**, which is essential for many applications to function correctly. The various IPC mechanisms that are available, and how developers can use them to produce reliable and effective applications.

IPC in Android refers to the transmission of messages and data between various processes. An Android process is a self-contained unit of execution, meaning it has its own set of resources and occupies its own memory space. For a number of reasons, including the following, it is essential for processes to be able to communicate with one another:

- **Modularity:** Android apps are made to be modular, which means they are made up of a number of parts that can be used in other apps. IPC enables interprocess communication (IPC) between components of an application, even when those components are running in different processes.
- **Security:** Because Android has a sandboxed architecture, each application is isolated and runs in its own environment. Applications can communicate with one another through IPC while still maintaining this isolation, which is essential for maintaining the system's security.
- **Performance:** The memory and processing power of Android devices are typically constrained. The system's overall performance can be enhanced by the resource sharing that IPC enables between various processes.

In Android, various IPC mechanisms are available, including the following:

- **Intents:** An application's various components or other applications can communicate with one another using intents, a messaging system. Intentions can be used to launch a service, a new activity, or a public event.
- **Content Providers:** Content providers are a way for various applications to share data. Access to a shared data store, such as a SQLite database, is controlled by the use of content providers.
- **Binders:** In Android, binders are a basic method of communication between various processes. The Android interface definition language (AIDL), which is used to define the interface between various processes, is implemented using binder.

These IPC mechanisms can be used by developers to build reliable and effective applications. For instance, they can use Content Providers to share data between various applications, Intents to start a new activity in a different process, or Binders to communicate between various processes. Developers must exercise caution when using IPC, though, as improper use of the technology can result in security flaws.

In conclusion, IPC is a crucial component of the Android operating system that enables communication between various processes. Intents, Content Providers, and Binders are just a few of the IPC mechanisms that Android offers developers in order to help them build robust and effective applications. Developers must exercise caution when using IPC, though, as improper use of the technology can result in security flaws. IPC enables the modular, safe, and effective design of Android applications, making it a crucial component of the Android operating system.

PROCESS SCHEDULING

The Android operating system makes use of process scheduling mechanisms, just as other operating systems. To execute the processes effectively, it **supports all scheduling approaches** such as **first in, first out, round robin scheduling, and scheduling techniques.** Different sorts of schedulers are now utilized to carry out scheduling strategies, for instance, some I/O schedulers are **FIFO** (first in first out), **CFQ** (full fair queue), **SIO** (simple input output to keep minimum number of overheads), etc.

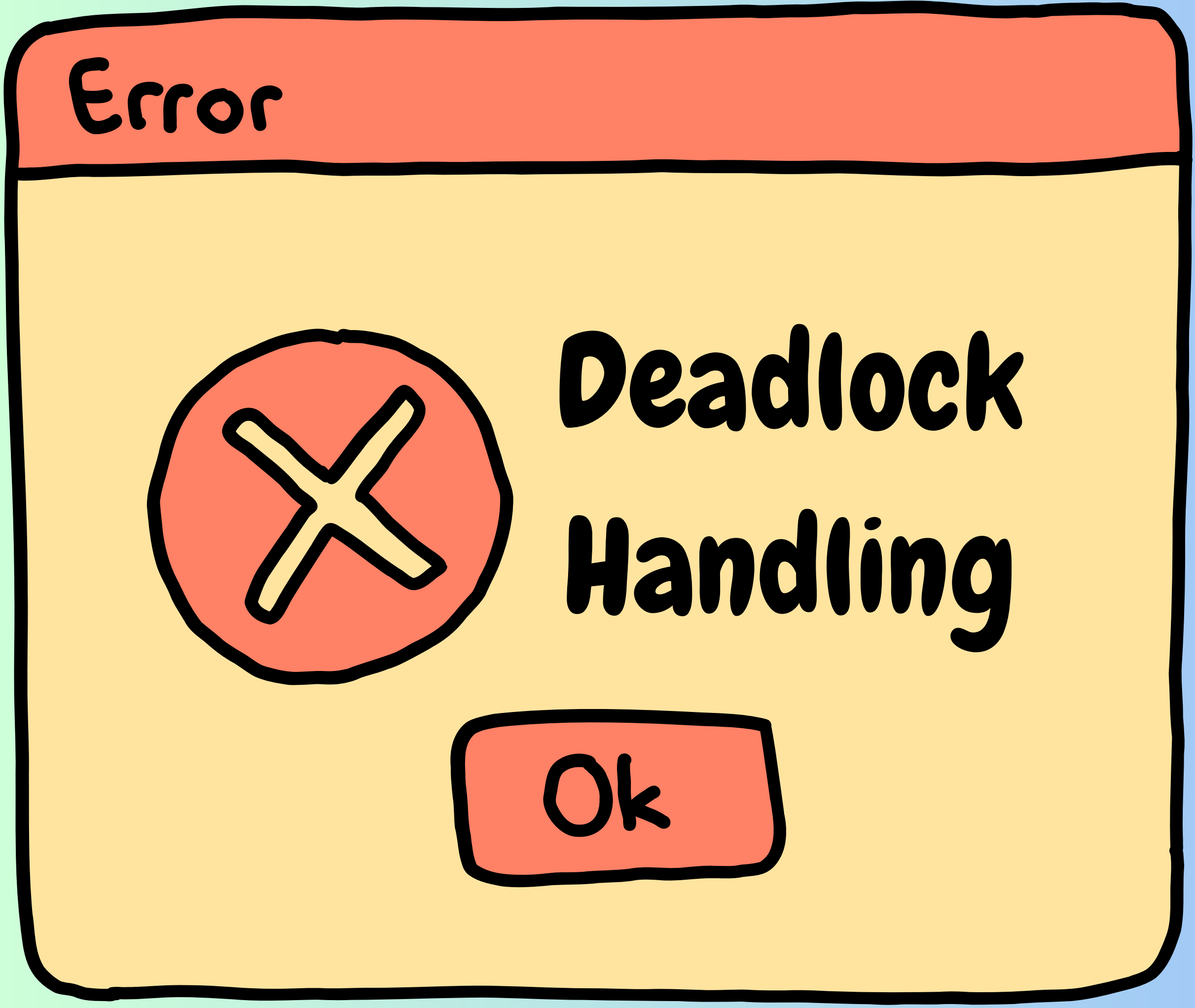
Preemptive Scheduling :

Task Scheduling is the fundamental idea in every operating system that enables the system to support multitasking, multiprocessing, etc. The main method by which various programs are permitted to share a single CPU is task scheduling. The software used to complete this task is the **scheduler** and **dispatcher**. Since the Linux Kernel is the foundation of the Android operating system, O(1) scheduling algorithm is used. No matter how many processes are running on the operating system, the scheduler utilized is called Completely Fair Scheduler since it can schedule processes in a fixed amount of time. When high priority jobs are in the queue, preemptive task scheduling includes interrupt the lower priority tasks. Due to the medium CPU utilisation, high turnaround and reaction times, and low CPU utilisation, this scheduling is specifically utilised for mobile operating systems.

Scheduling Techniques:
Processes on Android are divided into real time processes and ordinary processes, just like in the Linux operating system, as was previously discussed in the section "Process attributes and types." Real-time processes have a priority range of 0-99, while regular processes have a priority range of 100-139. The scheduling approaches used by the two processes also differ.
Different scheduler types are used to carry out the scheduling procedures, for instance, some I/O scheduler are **FIFO** (first in first out), **CFQ** (Complete Fair Queue), which shares the available I/O bandwidth equally, **SIO** (simple input output to keep a low number of overheads), etc.

For real time process:
For real-time processes in Android, the "**sched.FIFO**" policy is used. The first in, first out management principle is applied in this policy. If no real-time activity of greater priority is actively using the CPU at the time a process is occupying it, the process will continue to do so. This demonstrates that the Android operating system wants the real-time process to run continuously and with a high priority and does not want to halt execution because a time slice has run out. However, the technology won't permit the real-time procedure to operate continuously. The CPU-intensive process won't be designated as a real-time process. The I/O consuming process, which is more sensitive to real-time, is more likely to benefit from real-time processes.

For ordinary process:
The "**sched.other**" scheduling technique that Android's default process utilises gives it a nine-level priority. Depending on the circumstance, various strategies are used in this. According to its type, as previously mentioned, the procedure in another is given priority.



A deadlock occurs when **two or more processes are waiting endlessly for each other to release a resource they both require finishing their tasks**. In the case of an Android operating system, this can result in frozen or unresponsive applications as well as cause the entire system to halt. Android uses a variety of management systems to make sure processes can access resources without becoming stuck waiting for one another, preventing deadlocks from happening.

Use of locks and semaphores: is one of the main methods for managing deadlocks in Android. By using these tools, **processes are prevented from accessing resources that are already in use by another process**. By implementing a timeout period after which the process will be released from its waiting state and permitted to continue executing, locks and semaphores also prevent processes from waiting indefinitely for a resource.

The use of a hierarchy: of locks is another strategy employed by Android to avoid deadlocks. As a result, situations where multiple processes are waiting for one another to release locks that they require are avoided and processes can **acquire locks in a specific order**. To avoid a deadlock, the system will make sure that both processes acquire resource Y in the same order, for instance, if process A needs resource X and resource Y and process B needs resource Y and resource Z.

Android also employs a method known as **deadlock detection** to find and fix deadlocks before they happen. This involves checking the system on a regular basis for potential deadlock situations; if one is found, the system will act to resolve it. It might, for instance, end the process that is causing the deadlock or release a resource that it is holding for a process that is deadlocked.

To avoid processes waiting indefinitely for resources, Android uses a system of timeouts and retries. A process will be allowed to exit the waiting state and resume execution if it is unable to acquire a resource within a predetermined amount of time. The process then has the option of switching to an alternative resource that is available or trying to acquire the resource again later.

Concluding, Android employs a variety of methods to handle deadlocks and stop them from happening. The use of locks and semaphores, a hierarchy of locks, deadlock detection, timeouts, and retries are some of these methods. Android can access resources without being forced to wait for one another by using these techniques, making the operating system more dependable and stable.

MEMORY MANAGEMENT

Memory is managed by the **Dalvik virtual machine and the Android Runtime using paging and memory mapping**. As a result, any memory changes made by a program, whether through the allocation of new objects or touching mmapped pages, remain in RAM and cannot be paged out. To make memory accessible to the garbage collector, an app can only release object references that it has stored in memory. With one exception: **if the system needs to use RAM, any files that were mmapped in without alteration, such as code, can be paged out of RAM.**

Garbage Collection:

Each memory allocation is tracked by a controlled memory environment, like the **ART or Dalvik virtual machine**. Without the programmer's help, it returns unused memory to the heap once it has been determined that the application is no longer using it. Garbage collection is the **process used to recover unused memory in a controlled memory environment**. Finding data items in a programme that cannot be accessible in the future is one of the two objectives of garbage collection, together with recovering the resources consumed by those things.

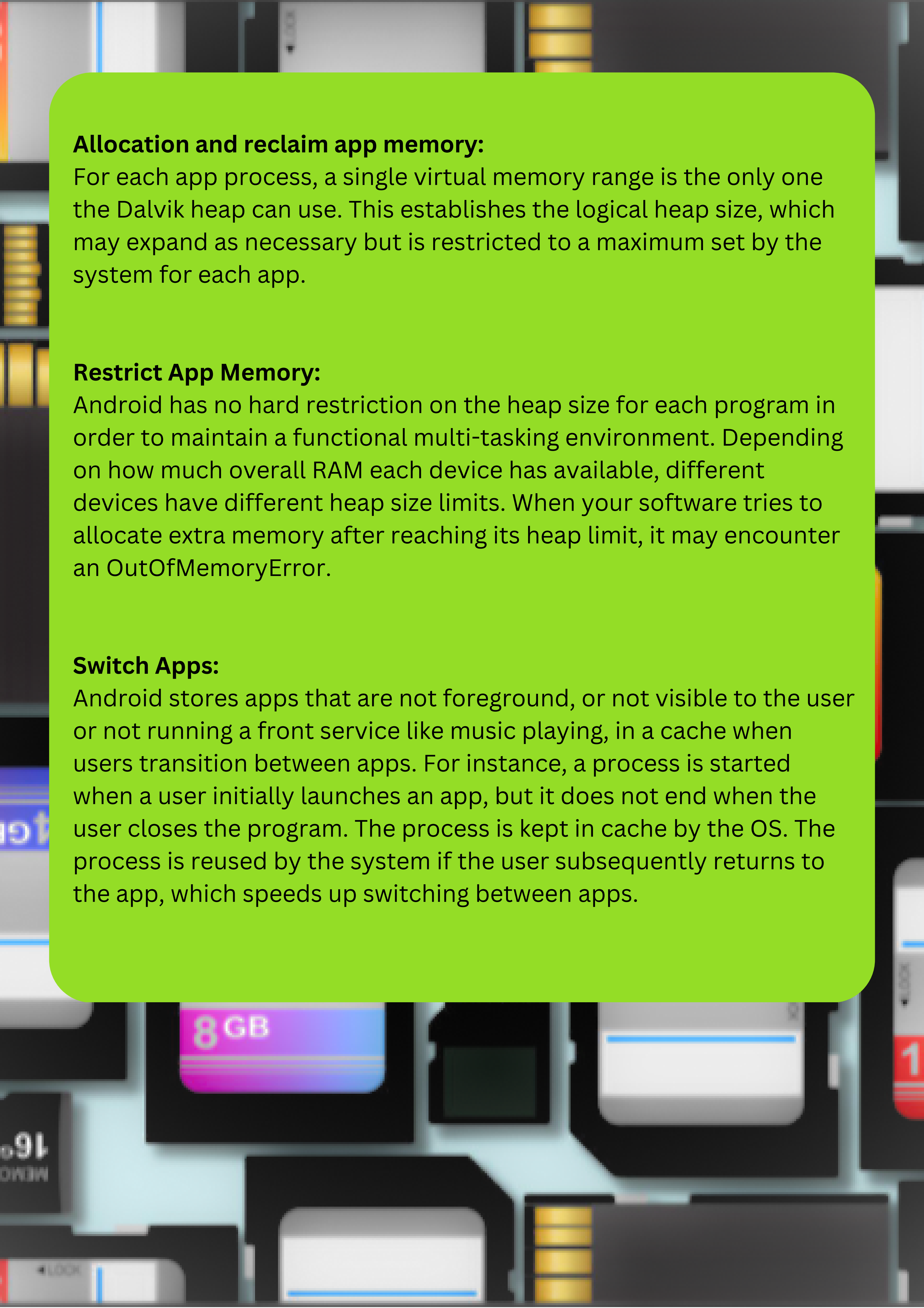
Shared Memory:

Android tries to share RAM pages among processes to accommodate what it needs in RAM. These are some of the ways it can achieve that:

1) Each app process is a **fork of the Zygote existing process**. When the machine boots, the Zygote process begins by loading shared framework code and resources (such activity themes).

2) **O processes are where most static data is mapped**. Data can be exchanged across processes using this method, as well as paged out when necessary.

3) In many instances, Android uses intentionally created **shared memory areas** (either with **ashmem or gralloc**) to share the same dynamic RAM across processes.



Allocation and reclaim app memory:
For each app process, a single virtual memory range is the only one the Dalvik heap can use. This establishes the logical heap size, which may expand as necessary but is restricted to a maximum set by the system for each app.

Restrict App Memory:
Android has no hard restriction on the heap size for each program in order to maintain a functional multi-tasking environment. Depending on how much overall RAM each device has available, different devices have different heap size limits. When your software tries to allocate extra memory after reaching its heap limit, it may encounter an OutOfMemoryError.

Switch Apps:
Android stores apps that are not foreground, or not visible to the user or not running a front service like music playing, in a cache when users transition between apps. For instance, a process is started when a user initially launches an app, but it does not end when the user closes the program. The process is kept in cache by the OS. The process is reused by the system if the user subsequently returns to the app, which speeds up switching between apps.

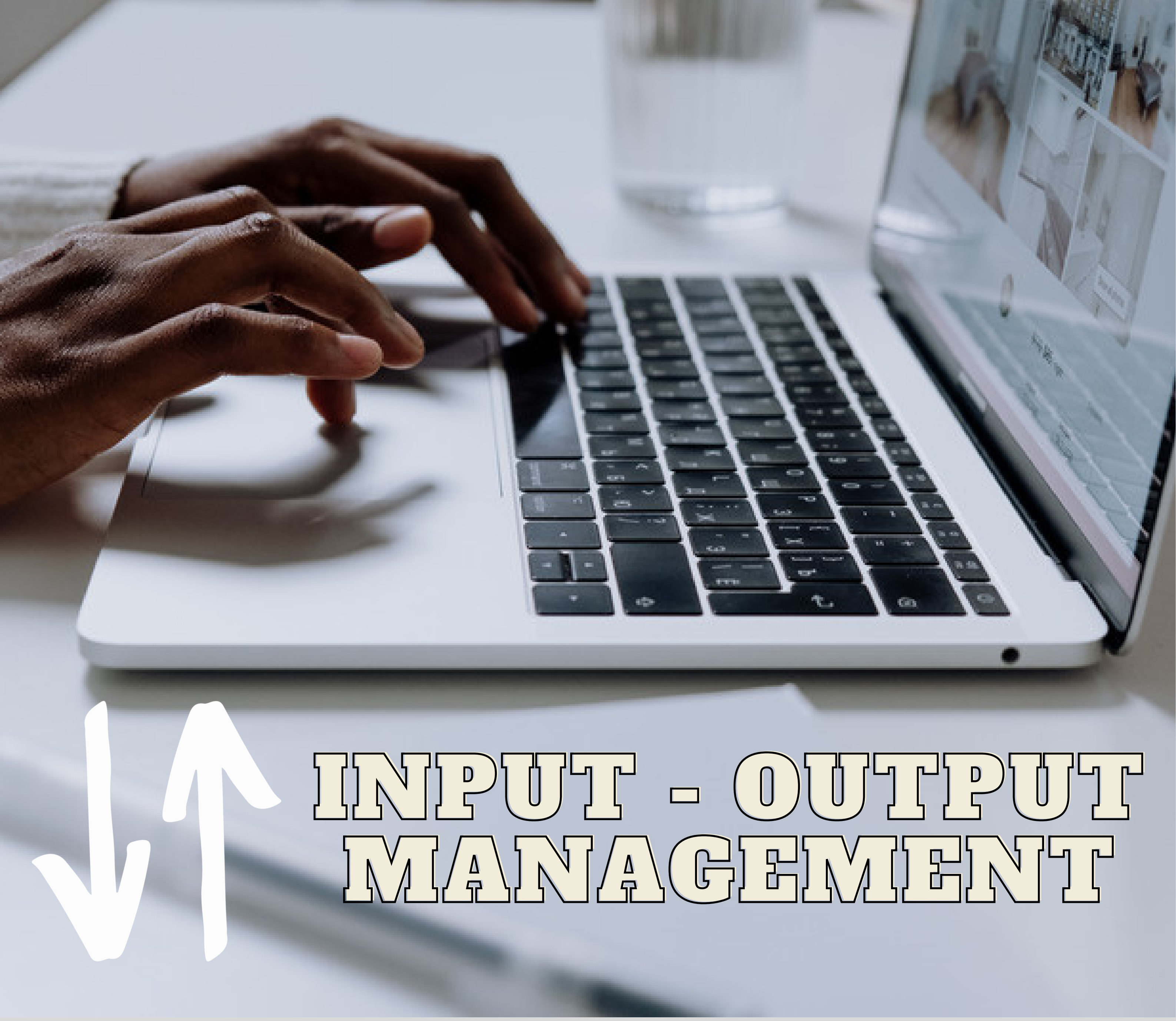
Storage Management

The OS is expected to manage the internal storage in two different ways. Check for occupied space first, then grant access to an app to read and write to internal storage. Apps on Android 10 and later can only access the device's media components. The majority of Android devices have **e-MMC storage**. They offer **read and write speeds of up to 300MB/s.**

Android's file system is similar to a Linux one with a **root directory**; you can see it in the image to the right. **Different partitions exist, each with their own features**. Only the OS itself is in charge of managing internal storage or memory cards.

The system's storage service is accessed through **Storage Manager**. **Opaque Binary Blobs (OBBs) are handled by the storage manager together with other storage-related things.**

OBBs have a filesystem that can be encrypted on the hard drive and mounted whenever an application needs it. OBBs, which can be several gigabytes in size, are a fantastic way to provide substantial amounts of binary assets without putting them into APKs. But because of their vastness, they are probably kept in a common storage pool that is open to all programs. **The OBB file's security is not guaranteed by the system**; if any program edits the O8B, there is no assurance that reading from that OBB will result in the desired output.



INPUT - OUTPUT MANAGEMENT

- Any operating system must have input/output (I/O), and Android is no different. **I/O is the management of the data flow to and from devices** like sensors, keyboards, and displays through communication with those devices.
- The Linux I/O system, which offers a unified interface for managing I/O operations, is built on top of the Android I/O system. This interface is expanded by the Android operating system to support a number of gadgets, such as touchscreens, cameras, and microphones. **Developers can access these devices and control their I/O operations using a set of Android APIs.**
- Android manages I/O devices using a **driver model**. With the help of this model, device makers can create and disseminate drivers that can be downloaded and installed on Android devices. The Android operating system can interact with devices and control their I/O functions thanks to these drivers.

- Additionally, Android **offers a collection of APIs** that let programmers access and control I/O devices. These APIs include the **Camera API**, which enables programmers to use the device's camera to take pictures and videos. While the **Sensor API** gives access to the device's various sensors, including the accelerometer and gyroscope, the **Audio API** enables developers to control audio playback and recording on the device.
- Support for external storage devices is another crucial feature of the Android I/O system. External storage options, like SD cards, are frequently available on Android devices and can be used to store files and data. **The Android operating system has APIs that let programmers read from and write data to external storage devices as well as access and manage these devices.**
- Additionally, **Android supports USB-connected devices**. Users can now connect USB devices like game controllers, mice, and keyboards to their Android phones and tablets. The **USB Host API** that comes with Android enables programmers to control USB devices that are connected to the device, including reading and writing data to and from these devices.
- The Android I/O system's support for input methods is one of its most important features. A number of input devices, such as touchscreens, keyboards, and voice recognition, are supported by Android. Developers can create unique input methods and incorporate them into the Android system thanks to the Input **Method Editor (IME) framework** that is part of the system.
- I/O operations on Android devices can be efficiently managed by users and developers thanks to the robust and adaptable Android I/O system. Its ability to support external storage, USB devices, and input techniques makes it a flexible system that can accommodate a variety of user requirements. **Developers can easily integrate with devices and efficiently manage I/O operations thanks to the use of a driver model and APIs**. The I/O system of the Android system is essential to its success as it continues to develop and get better.

CONCLUSION

Android is a new area in the world of computers. The effective use of technology has changed as a result. In this case study, we looked at the various components of the Android system from the perspective of process management. Even if the current Android OS version is designed for speed and excellent performance, it still has several problems that need to be fixed. The benefits and drawbacks of android are as follows:

Features:

- Android enables programmers to **create borderless applications**.
- Because Android is an open source project, **anyone can use it and adapt it to suit their needs**.
- Android is fast and offers **high performance**, and there are fewer overheads at the kernel thanks to the usage of a monolithic kernel.

Limitations:

- There are some restrictions in addition to the strong features provided, such as the **difficulty in preventing deadlocks despite our ability to detect them**. Another problem that stops app developers from limiting their capabilities is the **restriction on memory**.

In addition to the previously mentioned topics, Android also uses **memory management and power management**. The memory management is **page-based**, and the physical address space and logical address space are used to translate virtual addresses to physical addresses.

REFERENCES

- https://www.stackoverflow.com
- https://www.developer.android.com
- https://www.quora.com/Why-is-ART-better-than-Dalvik
- http://coltf.blogspot.in/p/android-os-processes-and-zygote.html
- https://cseweb.ucsd.edu/classes/fa10/cse120/lectures/CSE120-lecture.pdf/
- http://www.onsandroid.com/2014/10/in-depth-android-boot-sequence process.html Research and Development of Mobile Application for Android (Research Paper)
- Analysis on Process Code schedule of Android Dalvik Virtual Machine(Research Paper)
- Wikipedia