Case Study: Android OS

OPERATING SYSTEMS
CECSC09-I

Department of Computer Engineering

Submitted By:

ASHISH KUMAR 2019UCO1518
SANDEEP JAIN 2019UCO1522

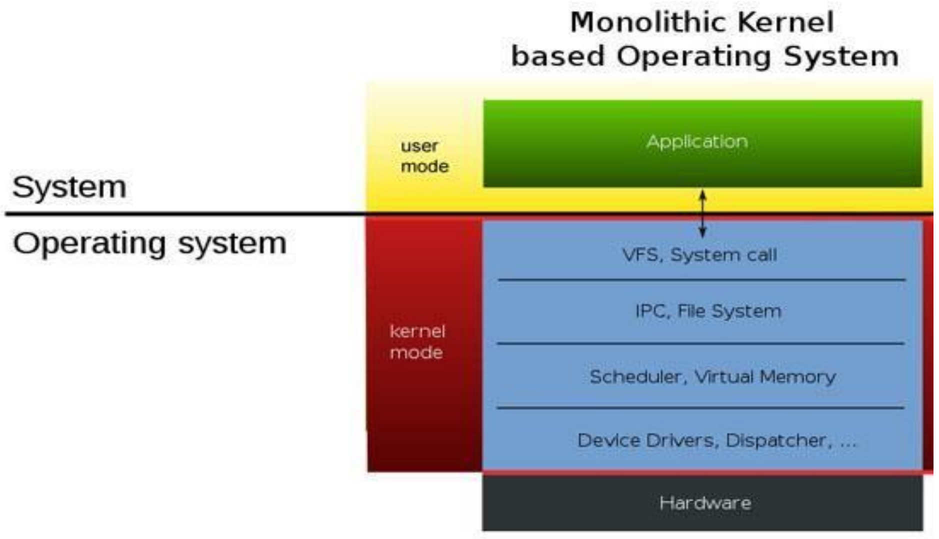
Netaji Subash University of Technology

INDEX

Introduction

Android is a software stack for mobile devices that includes an operating system, middleware and key applications, it is open source software and is developed by Google. The android operating system is based on the Linux platform which is based on the monolithic kernel which means that the complete operating system works in a single kernel space but this approach has a drawback that since everything is present in the kernel space so the size of the kernel increases, so in order to avoid this problem the android uses the modern approach of monolithic kernel where the different modules like file systems, device drivers, executable formats etc. present in the kernel space are loaded and unloaded dynamically at the runtime, this modern approach helps in the better use of resources like small size of kernel, better maintainability of the code.

- The monolithic kernel is different from the microkernel in the sense that it involves everything inside the kernel space while the microkernel kept only the essential things in the kernel and put all the non-essential things like files system, drivers etc. in user space.
- The key features of monolithic kernel are its speed of execution due to fewer overheads whereas that of the microkernel is that it is more flexible, crash resistant and more secure.
- All the advantages of monolithic kernel are the limitations of the microkernel and viceversa.
- We can also say that android OS has modular monolithic kernel.


Monolithic Kernel based Operating System
user mode
Application
System
Operating system
kernel mode
VFS, System call
IPC, File System
Scheduler, Virtual Memory
Device Drivers, Dispatcher, ...
Hardware

Android Platform Architecture

The architecture of the android operating system is mainly divided into six sub sections.

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

1. <u>Linux Kernel</u>: This is the foundation of the android platform. Basically a kernel is a program that continuously runs in the system. The Linux kernel is the bottom most layer of the android architecture. The current version of android runs on Linux version 4.4. The Linux kernel provides support for memory management, security management, process management and device management. It also consists of a list of device drivers that are used to communicate with other peripheral devices. A device driver is software that provide software interface to the hardware devices. The android runtime relies on the Linux kernel for underlying functionalities such as multi-threading and low-level memory management.

2. <u>Hardware Abstraction Layer</u>: The Hardware abstraction layer basically bridges the gap between the hardware and software. It provides standard interfaces that expose device hardware capabilities to the higher level java API framework. It is a layer of programming that allows a computer OS to interact with a hardware device at a general or abstract level rather than at a detailed hardware level. HAL can be called from either the OS's kernel or from a device driver. In either case, the calling program can interact with the device in a more general way than it would otherwise.

3. <u>Android Runtime:</u> The android runtime consists of a collection of the core android libraries and the Dalvik virtual machine (although the android versions after Kit Kat uses the more optimized Android runtime(ART)).The android runtime is used to run multiple virtual machines on low-memory devices and its main features are better debugging support and garbage collection.

- The ART is more optimized than the Dalvik virtual machine because the DVM uses the JIT (just in time) compiler technology meaning that every time the code gets compiled to the native code during the execution of the application each time when the application run whereas ART uses AOT (ahead of time) technology in which the apps are compiled to the native code once during the installation process. Since thecode is compiled once so it makes the ART better than DVM.

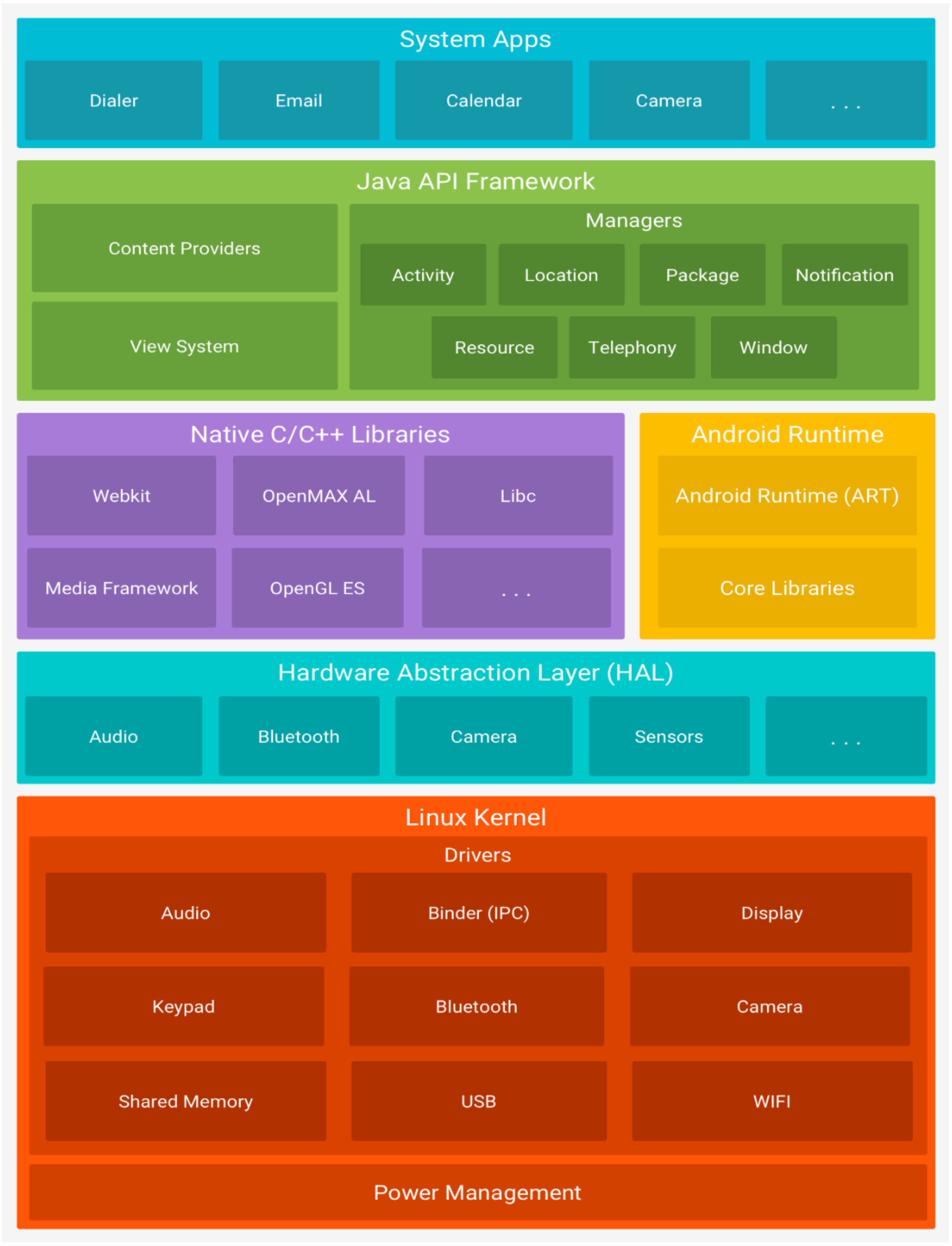
- The advantages of ART are that it improves speed (app start-up time), improved garbage collection and reduced memory footprints i.e. less referencing of memoryby an application.

4. Native C/C++ libraries : The native c/c++ libraries are used as supporting mechanism for many components and services like android runtime, hardware abstraction layer and other components that require direct access of these libraries. The e.g. of these libraries are OpenGL (for 3D graphics rendering), SQLite (for database management), web kit (for html content), SSL (for internet security).

5. Java API framework: All the features of the android OS are available to the user and developers with the help of Java API that act as a building block to create the android apps. The java API framework provides number of components and services like resource managers, notification managers, activity manager and content providers etc. which are helpful in building rich and modular applications.

6. System Apps: This section include the all the applications that are present in the android devices either they come with the android OS preinstalled or the user manually installs them with functionalities ranging from a simple calculator to a complex maps application.

The below diagram show the sub sections of android architecture:



Process and Thread Management

Introduction

A process in operating system is an instance of the operating system currently being executed.So the process management is a way by which different process share resources and execute in an optimal way so that properties like synchronization, mutual exclusion are maintained.

As the android operating system is based on the Linux kernel so it inherits many features of the Linux but it also deviates from Linux architecture in the sense that the it has a layer of abstraction called HAL which we have discussed earlier but the normal Linux distribution doesnot have that layer and also the size of the processor, battery consumption and other resourcesare not that much powerful so certain mechanism like OOM(out of memory killer) ,ART (android runtime) etc. are used to make it fast and reliable.

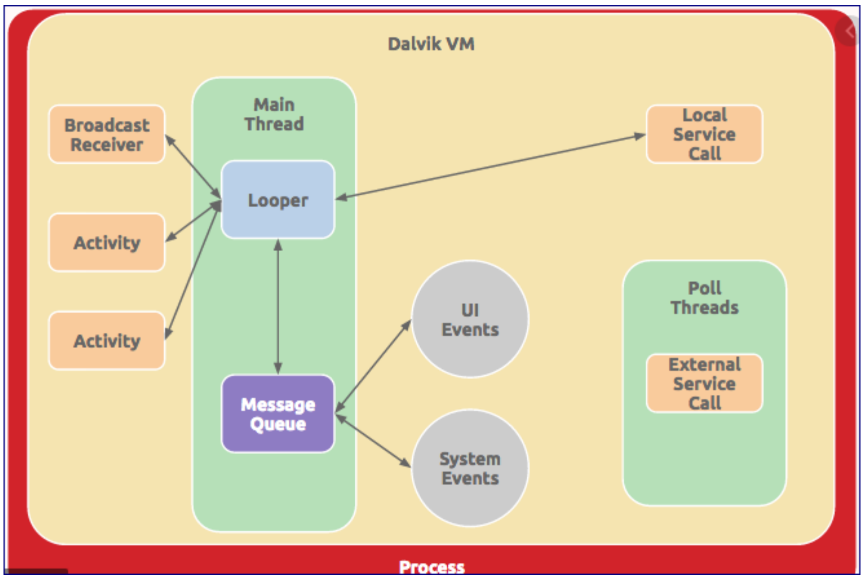
The android uses java as its primary language for running the android applications but insteadof using the Java virtual machine it uses the Dalvik virtual machine. All the processes in androidare executed with the help of Dalvik Virtual Machine. The Dalvik is a virtual machine where every android application runs. The following key points are associated with the DVM:

- The Dalvik VM uses the .dex file (dalvik executable) which is smaller and light weight insize than the java .jar file. The .dex file is very important for applications where memory and the power consumption have a limitation.
- The .dex file is generated from the java .class files also it is native in nature and processof its creation is as follows:

$$\begin{tikzcd}\text{.java files} \arrow[r, "\text{(Java compiler)}"] & \text{.class files} \arrow[r, "\text{(Dx tool)}"] & \text{.dex file}\end{tikzcd}$$

- The dalvik virtual machine is a register-based architecture which ensures the memory management.
- Each newly created process in android has its own VM instance and therefore has its own process space, by doing this the DVM ensure the security of the applications i.e. ifat any moment one instance of VM fails then it does not affect the execution of other processes in the system.
- The DVM is dependent on the Linux kernel and each DVM process is Linux process andeach DVM thread is a Linux thread.

- The Dalvik uses the Zygote process model and is similar to the fork process of Linux. The below figure shows the mapping of application processes to the Linux kernel using DVM:



When the android operating system first starts then it starts the boot loader which does thing like initializing cache, initializing kernel etc. for execution. Then the first process that gets created is the init process .The init process is the parent of all the child processes that will be created at later point of time. It does two things first it create the low level Linux processes called as daemons i.e. the process that are running in background and have not interaction with the user (in windows operating system we call these as services). These daemons handle low level hardware interfaces, second it creates the zygote process. The zygote is also a type of daemon whose mission is to launch the applications, so we can say that zygote is the child process of init process and parent of all the other processes. When init process launches zygote, it creates the first Dalvik VM and calls zygote's main() method, after that it preloads allthe necessary java classes and resources so that each new process can use those resources instead of creating new resources copies each time. Then the zygote start listening to new processes and whenever a new application comes it forks() a process and create a VM instance. Also each application has its own process space and two applications doesn't share a commonprocess.

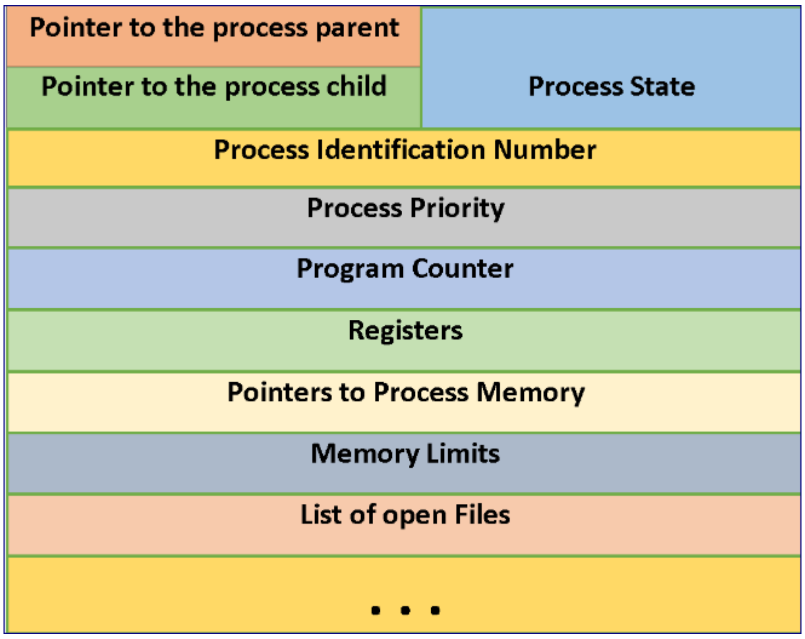
After the forking of the child processes the child process then starts the app components like activities, services, content providers and broad receivers depending on the types of the operation.

- Activities: Activities are the entry point for interacting with the user. These are the single screen with a user interface.
- Services: It is a component that runs in the background to perform long-running operations or to perform work for remote processes.
- Broadcast receivers: These are the components through which a system enables certain events. These are sleeping all the time in the system and whenever an event gets triggered then all of them wake up and one of them does the task depending on event_id.
- Content providers: These components are used for data transfer between apps or between app and file storage in the system.

Attributes of process

The process management in android is similar to the process management in other operating systems i.e. it also have a process control block ,a life cycle of process and otherthings like process terminations.

The process control block in android is shown as:


Pointer to the process parent
Pointer to the process child
Process State
Process Identification Number
Process Priority
Program Counter
Registers
Pointers to Process Memory
Memory Limits
List of open Files
. . .

Process Control Block

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 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 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.

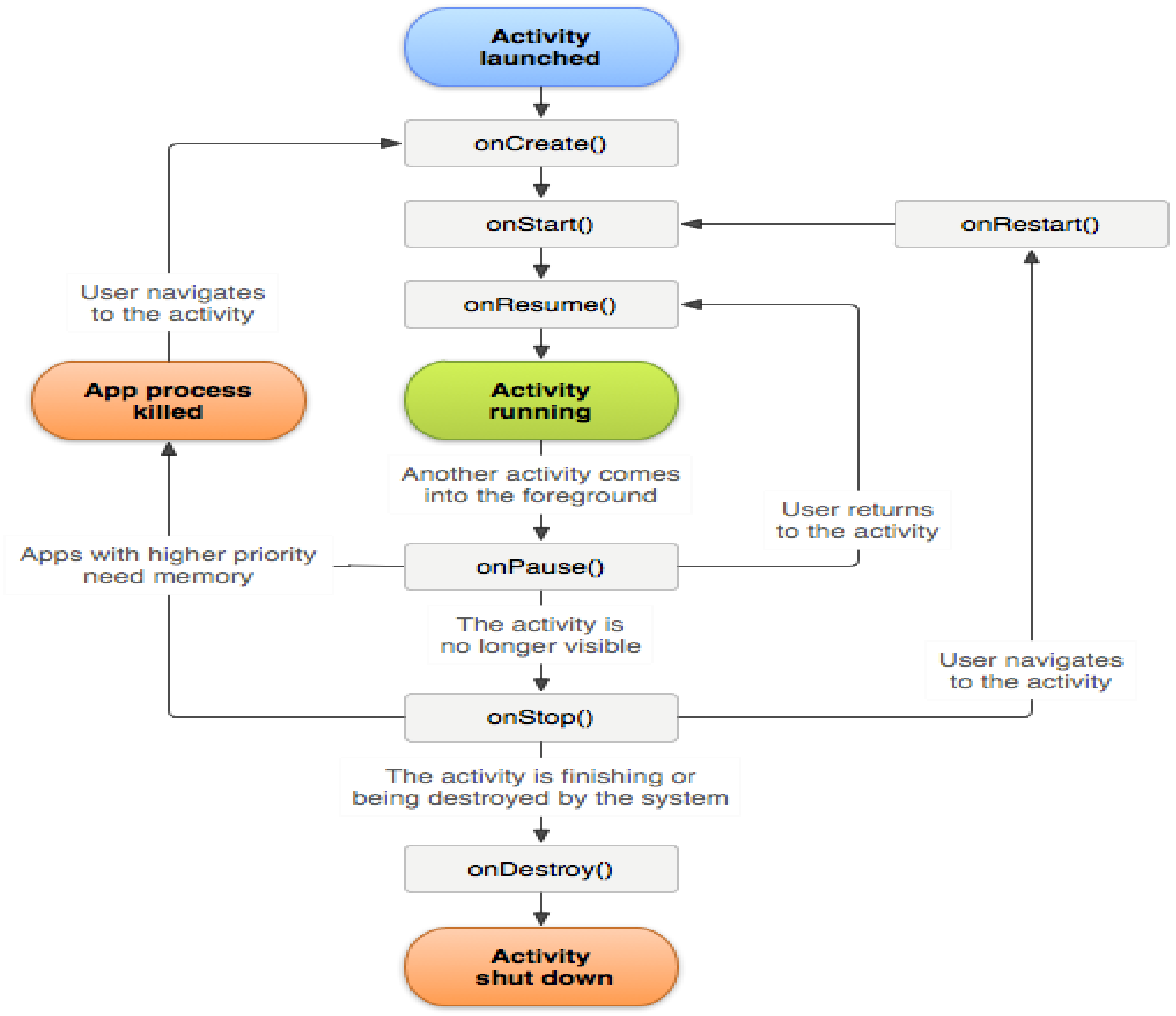
- Paused: If an activity has lost focus but is still visible then it is in the pause state, a paused activity is alive, but can be killed by the system in extreme low memory situation by the OOM.

- Stopped: The activity is completely stopped either the execution has been completed or the OS require 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 very useful step because android is and OS with low memory and also low computation power.

The activity is very important because each process has several number of activities so the process life cycle is depends heavily on the activity life cycle. In the modern android versions like Nougat, Marshmallow etc. the fragments are preferred over the activities. The fragments are basically light weight activities that are generally reusable in nature.

The pictorial representation of the activity life cycle is:


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Now there are several types of limitations on the android operating system because of its small size. So in order to use the available resources in the most optimized manner the different processes present in the OS are subdivided on the basis of their priority and they form an “importance hierarchy” similar to the states hierarchy of an activity.

The process types are as follows:

- Foreground process: This is the process responsible when the user is interacting with the screen. This is given the top priority. A process is said to be foreground if it has a running activity at the top of screen with which the user is interacting, a broadcast receiver and a service. These are few in number and are only killed when the memory is very low so that these need to be killed to make the user interface responsive.

- Visible process: A process is said to be visible when it is running in foreground but the user is not currently interacting with the process. An e.g. of this type of process is whenthe onpause () method is called and the process gets paused.

- Service process: A service process the one that is holding a service started with the startService () method .Though these processes are not visible to the user but they havean impact on the user .These process include things like uploading or downloading something. These have generally low priority than the foreground and visible process. If these processes are running for very long time then sometimes these are also demoted so as to avoid memory leaks and high RAM consumption.

- Background process: Background processes are not currently visible to the user. They have no impact on the experience of using the phone. At any given time, many background processes are currently running. We can think of these background processes as “paused” apps. They’re kept in memory so you can quickly resume using them when you go back to them, but they aren’t using valuable CPU time or other non- memory resources.

- Cached process: These are the process that are currently not needed, so the system isfree to kill it as desired when memory is needed elsewhere. Generally these are the processes involved in memory management.

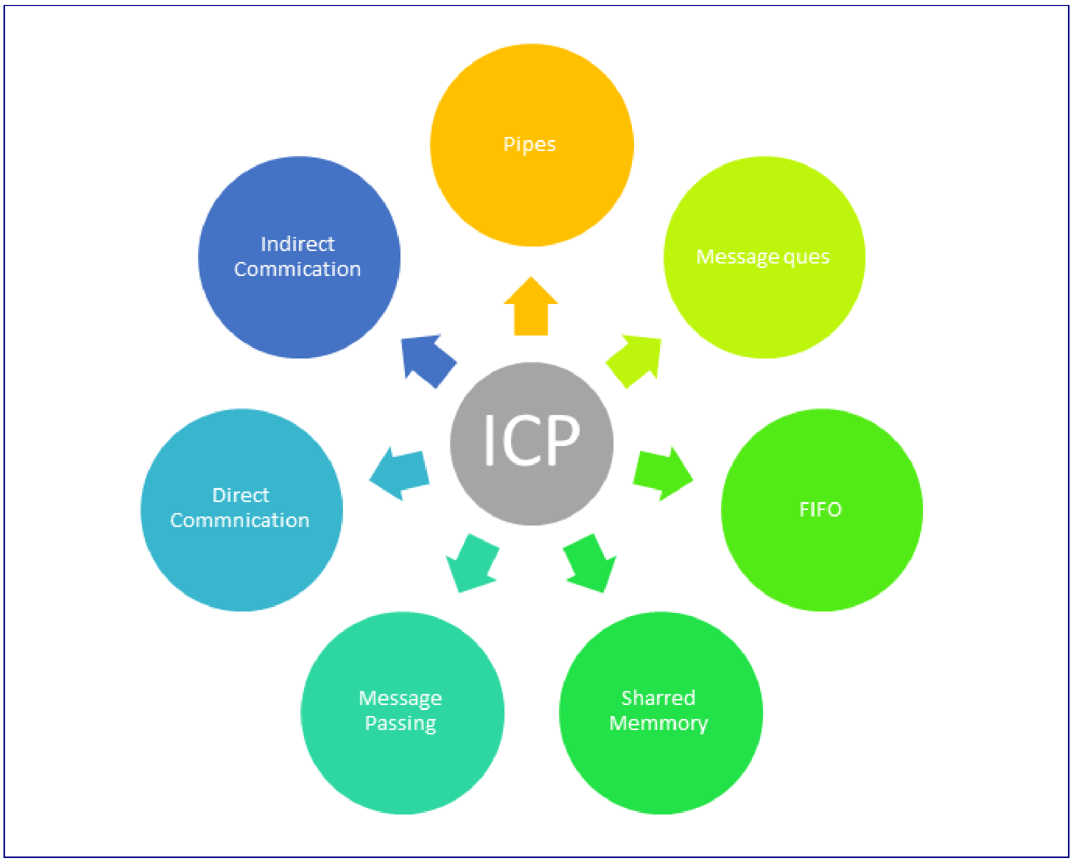
The priority of a process can also be increased and decreased as when needed for betterexecution of the applications.

Inter Process Communication

The inter process communication (IPC) describes the mechanism of how different types ofandroid components/process communicate with each other.

- Intents are messages which components can send and receive. It is universal mechanism of passing data between processes. With help of the intents we can start services or activities, invoke broadcast receivers and so on. These are an abstract description of an operation to be performed.
- Bundles are entities through which the data is passed. It is similar to the serialization ofan object (serialization is a process of converting an object into stream of bytes), but much faster on android. We can also get Bundles from intent via getExtras () method.

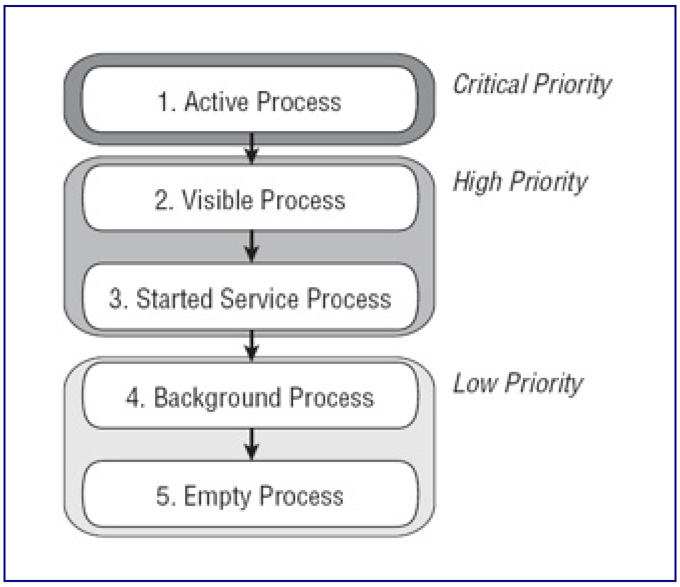
- Binders are the entities which allows activities and services obtain a reference to another services, it allows not simply send messages to services but directly invoke methods on them. In simple terms the binder are used to communicate between a server process and an application process and invoke methods of other process.



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 andscheduling 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 example
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.



Pre-emptive task scheduling - The key concept present in any operating system which allows the system to support multitasking, multiprocessing, etc. is Task Scheduling. Task Scheduling is the core which refers to the way the different processes are allowed to share the common CPU. Scheduler and dispatcher are the softwares which help to carry out this assignment. Android operating system uses O(1) scheduling algorithm as it is based on Linux Kernel. The scheduler used is named as Completely Fair Scheduler as the processes can schedule within a constant amount of time, regardless of how many processes are running on the operating system. Pre- emptive task scheduling involves interrupting the low priority tasks when high priority tasks are present in the queue. This scheduling is particularly used for mobile operating system as the CPU utilization is medium, turnaround time and response time is high. Mobile phones are required to meet specific time deadlines for the tasks to occur.

Scheduling techniques: As mentioned before in the section "Process attributes and types", process in Android like in the Linux system are divided into real-time processes and ordinary processes. The priority range of real-time processes is 0-99, and that of ordinary processes is 100-139. Moreover, the scheduling strategies of the two processes are also different.

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.

For Real time processes

In Android, "sched.FIFO" policy is used for real-time processes. This policy uses the first in, first out management rule. When a process occupies the CPU, if there is no real-time process with higher priority occupying or actively giving up, the process will keep using the CPU. This also shows that Android system wants the real-time process to run continuously with high priority, and does not want to interrupt execution due to the exhaustion of time slice. But the system will not let the real-time process run all the time. The CPU consuming process will not be set as the real- time process. The real-time process is more inclined to serve the I/O consuming process which is more sensitive to the real-time.

For Ordinary processes

The normal process in Android uses "sched.other" scheduling strategy, which prioritizes the normal process to 9 levels. In this different type of techniques are employed depending upon the situation. For the process in other the priority is given to the process based on it type as mentioned before.

Process Termination

After the process has been executed it is to be terminated by the OS. The process can be killedin two ways:

- An application can call a method to kill processes it has permission to kill. This means that if the process is not the part of the same application, it can't kill other processes. We can also provide permission to an application to kill processes of other applications.
- The android OS has a LRU queue which uses the page swapping technology of Linux and keep tracks of the application that are not been

used. So in case when the androidis low on memory then the Out-of-memory killer kills the cached process and other processes based on the priority so as to make the user interface responsive.

The process is generally killed using the Process. Kill Process (int,pid) method where pid is the process id of the process to bekilled.

To determine which processes should be killed when low on memory, Android places each process into an "importance hierarchy" based on the components running in them and the state of those components. These process types are (in order ofimportance):

<u>A foreground process</u> is one that is required for what the user is currently doing. Various application components can cause its containing process to be considered foreground in different ways. A process is considered to be in the foreground if any of the following conditions hold:

1. It is running an Activity at the top of the screen that theuser is interacting with (its onResume() method has been called).
2. It has a BroadcastReceiver that is currently running (its BroadcastReceiver.onReceive() method is executing).
3. It has a Service that is currently executing code in one of its callbacks (Service.onCreate(), Service.onStart(), or Service.onDestroy()).
4. . There will only ever be a few such processes in the system,and these will only be killed as a last resort if memory is so low that not even these processes can continue to run.

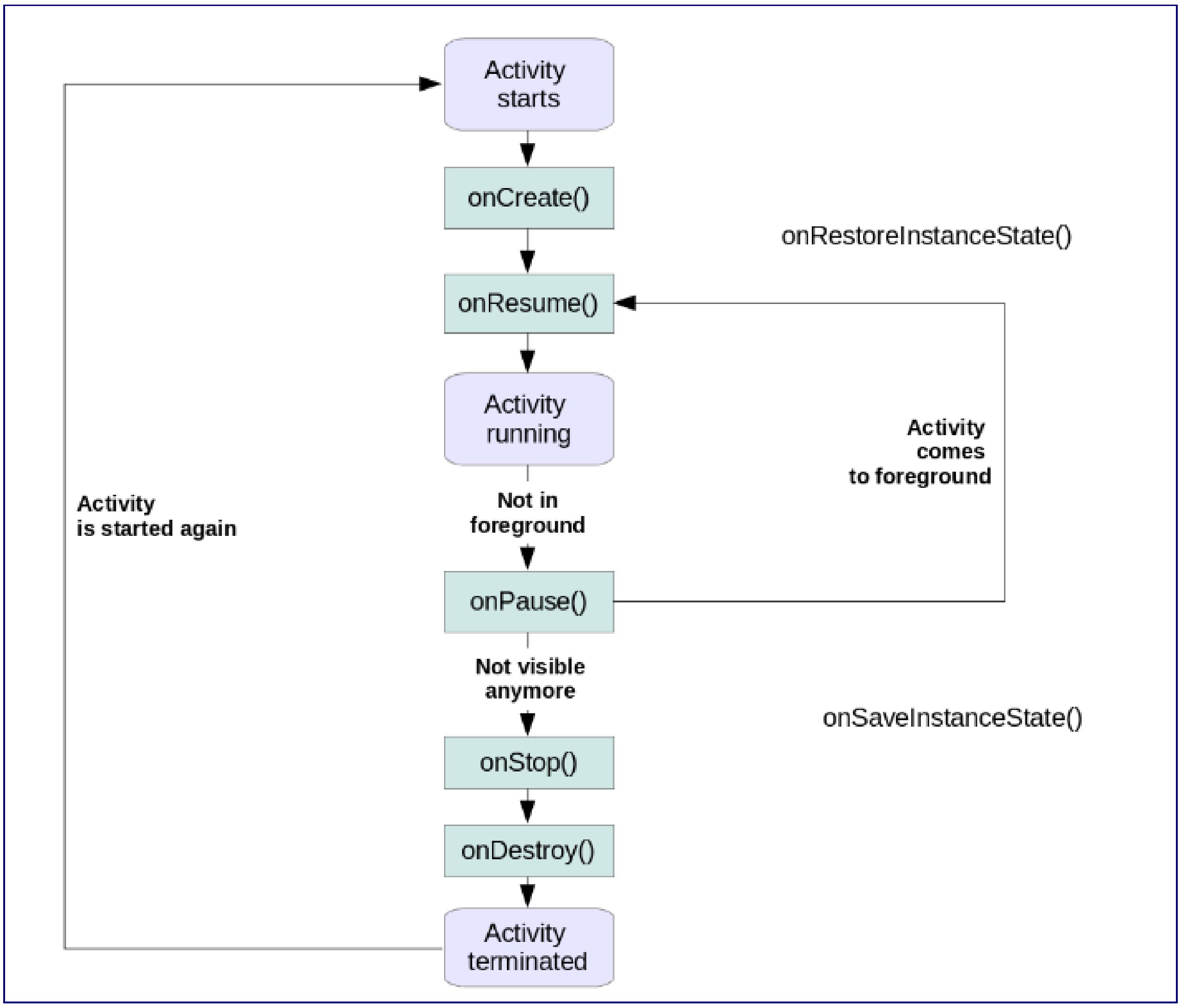
Generally, at this point, the device has reached a memory paging state, so this action is required in order to keep theuser interface responsive.

<u>A visible process</u> is doing work that the user is currently aware of, so killing it would have a noticeable negative impact on the user experience. A process is considered visible in the following conditions:

1. It is running an Activity that is visible to the user on- screen but not in the foreground (its onPause() method has been called). This may occur, for example, if the foreground Activity is displayed as a dialog that allows the previous Activity to be seen behind it.
2. It has a Service that is running as a foreground service, through Service.startForeground() (which is asking thesystem to treat the service as something the user is aware of or essentially visible to them).
3. It is hosting a service that the system is using for a particular feature that the user is aware, such as livewallpaper, input method service, etc.

The number of these processes running in the system is less bounded than foreground processes, but still relatively controlled. These processes are considered extremely important and will not be killed unless doing so is

Required to keep all foreground processes running.



Threads in Android

The threads are also an important part of the android OS. Each process has a separate thread by default. When an application is launched, the system creates a thread of execution for the app and is known as the main thread. This thread is very important because it is in charge of dispatching events including drawing events. The main thread is also 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 toeach component are dispatched for that thread.

Now there are situations when an application performs intensive work in response to the user interaction, the single thread model i.e. the UI thread performs poor and we does not get the desired output, also the UI thread is not thread-safe, so we use other type of thread called as worker threads who work along with the main UI thread. However there are some limitations on the worker threads as:

- The worker threads can’t block the UI thread.
- The worker threads only run in the background and can’t update the application UI, the application UI can be updated by only the main thread.

The threads in android are generally implemented with the help of pthread libraries. The pthread (POSIX thread) libraries are natively implemented in C/C++.

Every thread has a name for identification purposes. More than one thread may have the same name. If a name is not specified when a thread is created then a new name is generated for it.

Process Synchronization

There are sometimes situations when two or more processes wants 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 from 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.

Interrupts I/O

An alternative scheme for dealing with I/O is the interrupt-driven method. An interrupt is a signal to the microprocessor from a device that requires attention.

A device controller puts an interrupt signal on the bus when it needs CPU’s attention when CPU receives an interrupt, It saves its current state and invokes the appropriate interrupt handler using the interrupt vector (addresses of OS routines to handle various events).When the interrupting device has been dealt with, the CPU continues with its original task as if it had never been interrupted.

I/O software is often organized in the following layers –

1. User Level Libraries – This provides a simple interface to the user program to perform input and output. For example, stdio is a library provided by C and C++ programming languages.
2. Kernel Level Modules – This provides device drivers to interact with the device controller and device independent I/O modules used by the device drivers.
3. Hardware – This layer includes actual hardware and hardware controllers which interact with the device drivers and makes hardware alive.

A key concept in the design of I/O software is that it should be device independent where it should be possible to write programs that can access any I/O device without having to specify the device in advance. For example, a program that reads a file as input should be able to read a file on a floppy disk, on a hard disk, or on a CD-ROM, without having to modify the program for each different device.

Device Drivers

Device drivers are software modules that can be plugged into an OS to handle a particular device. The Operating System takes help from device drivers to handle all I/O devices.

A device driver performs the following jobs –

1) To accept requests from the device independent software above to it.
2) Interact with the device controller to take and give I/O and perform required error handling
3) Making sure that the request is executed successfully Interrupt handlers

An interrupt handler, also known as an interrupt service routine or ISR, is a piece of software or more specifically a callback function in an operating system or more specifically in a device driver, whose execution is triggered by the reception of an interrupt.

Device-Independent I/O Software

The basic function of the device-independent software is to perform the I/O functions that are common to all devices and to provide a uniform interface to the user-level software. Though it is difficult to write completely device independent software, we can write some modules which are common among all the devices.

Following is a list of functions of device-independent I/O Software –

1. Uniform interfacing for device drivers
2. Device naming - Mnemonic names mapped to major and minor device numbers

3. Device protection
4. Providing a device-independent block size
5. Buffering because data coming off a device cannot be stored in the final destination.
6. Storage allocation on block devices
7. Allocation and releasing dedicated devices
8. Error Reporting

<u>User-Space I/O Software</u>

These are the libraries which provide a richer and simplified interface to access the functionality of the kernel or ultimately interactive with the device drivers. Most of the user-level I/O software consists of library procedures with some exceptions like spooling system which is a way of dealing with dedicated I/O devices in a multiprogramming system.

<u>Kernel I/O Subsystem</u>

Kernel I/O Subsystem is responsible to provide many services related to I/O. Following are some of the services provided.

1. <u>Scheduling</u> − Kernel schedules a set of I/O requests to determine a good order in which to execute them. When an application issues a blocking I/O system call, the request is placed on the queue for that device. The Kernel I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by the applications.
2. <u>Buffering</u> − Kernel I/O Subsystem maintains a memory area known as a buffer that stores data while they are transferred between two devices or between devices with an application operation. Buffering is done to cope with a speed mismatch between the producer and consumer of a data stream or to adapt between devices that have different data transfer sizes.
3. <u>Caching</u> − Kernel maintains cache memory which is a region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original.
4. <u>Spooling and Device Reservation</u> − A spool is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. The spooling system copies the queued spool files to the printer one at a time. In some operating systems, spooling is managed by a system daemon process. In other operating systems, it is handled by an in kernel thread.
5. <u>Error Handling</u> − An operating system that uses protected memory can guard against many kinds of hardware and application errors.
6. <u>Main memory and the I/O device</u> - The CPU is only involved at the beginning and end of the transfer and interrupted only after the entire

block has been transferred.

Conclusion

Android is an emerging field of the computing devices. It has changed the way of using the technology in efficient manner. In this case study we have studied the architecture of android system and the view different aspects from the process management view point. Although the current version of android OS is optimized for speed and high performance but it has some issues that are still to be resolved. So the advantages and limitations of android as follows :

Features:

- Android allows developers to develop apps that have no borders.
- Android is an open source project means it is free so anybody can modify it and use itfor their purpose.
- The use of monolithic kernel in android makes it fast and provides high performanceand there are less overheads at the kernel

Limitations:

- Besides providing good features there are certain limitations like we can detect the deadlock but its prevention is hard. Also the limitation on less memory is also an issue which prevents the app manufacturers to restrict their capabilities.

Along with the above discussed topics android also employs memory management and powermanagement. The memory management is page-based memory management in which there is a virtual address to physical address mapping using logic address space and physical address space.

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