Case Study on Android

1. What is Android operating system?

The Android OS was originally created by Android, Inc., which was bought by Google in 2005. Google teamed up with other companies to form the Open Handset Alliance (OHA), which has become responsible for the continued development of the Android OS.

Android's underlying kernel is based on Linux, but it has been customized to suit Google's directions. There is no support for the GNU libraries and it does not have a native X Windows system. Inside the Linux kernel are found drivers for the display, camera, flash memory, keypad, WiFi and audio. The Linux kernel serves as an abstraction between the hardware and the rest of the software on the phone. It also takes care of core system services like security, memory management, process management and the network stack.

The Android OS is designed for phones. Its many features include:

- Integrated browser, based on the open source WebKit engine
- Optimized 2D and 3D graphics, multimedia and GSM connectivity
- Bluetooth
- EDGE
- 3G
- WiFi
- SQLite
- Camera
- GPS
- Compass
- Accelerometer

Software developers who want to create applications for the Android OS can download the Android Software Development Kit (SDK) for a specific version. The SDK includes a debugger, libraries, an emulator, some documentation, sample code and tutorials. For faster development, interested parties can use graphical integrated development environments (IDEs) such as Eclipse to write applications in Java.

Android story

- Android Inc was founded in Palo Alto, California,United States by Andy Rubin,Rich Miner,Nick Sears & Chris White -- Oct 2003
- Google acquired Android Inc -Aug2005
- The Open Handset Alliance, a consortium of several companies was formed -5thNov2007
- Android beta SDK released -12thNov2007

Case Study on Android

Android Versions:

Android Version 2.x.x – Gingerbread (6th December, 2010)

The Android Version Gingerbread brought a revolution into the world of mobile communication. Since the release of the first Android Version, Android had been trying to make its way to the core of the mobility market but their journey actually started with the release of this Android Version.

- Google added an intelligent User Interface into this particular Android Version
- New and improved keyboard the for the ease of the uses
- Added the feature of copy/paste
- Power Efficiency for the efficient use of the mobile battery
- Social Network related features added
- NFC or Near Field Communication Support added
- Video Call Support

Besides, the above features that were the added onto the Android Version there were also the ones that were added for the ease of developers. Basically, all the necessary things that a developer would require for developing anything and everything related to the android platform were added into this Android Version.

Android Version 3.x.x – Honeycomb (22nd February, 2011)

At the time of the release of this Android Version, tablets devices were getting famous in the market, so Android rather took a risky turn which did not go in their favor. Honeycomb was basically for the purpose of enriching the tablet UI. The list of features added are as below

- Multi Core Support to improve processing
- Tablet Support
- 3D UI Updated
  - Customizable home screens
  - Recent applications view
  - New Keyboard layout
- Media Transfer Protocol
- Google Talk video Chat
- Private Browsing for privacy improvement
- HTTP live streaming

A version update of this Android Version was released in the year 2012 that brought the feature of “Pay as you go” but since the Android Version was not as famous as it should have been, it did not go viral.

Android Version 4.0.x – Ice Cream Sandwich (18th October, 2011)

After the not so popular Android Version, codenamed Honeycomb, came the Ice Cream Sandwich which continued with the popularity of the Android Operating System. It came with several bug fixes and a large list of features were also added into the Android Operating System. The Features are as follows:

- New Lock Screen Actions
- Improved text input and spell-checking
- Control of the Network data
- Email app support
- Wi-Fi direct
- Bluetooth health device profile
- Social Stream to keep the contacts updated
- Video Stabilization
- QVGA video resolution API access
- Calendar provider updates
- Smoother screen rotation

Android Version 4.1.x – 4.3.x – Jelly Bean (9th July, 2012)

By the time of the release of Android Versions 4.0.x, codenamed Ice Cream Sandwich. Android had pulled away most of the users from the competitors and Google totally knew about what to changes to make in their next Android Version. This step was a crucial one because either people would move away from Android or stick to it for as long as it exists.

- Google Now
- Voice Search
- Speed Enhancements
- Camera app improvements
- External Keyboards and Gesture mode – improving accessibility
- Lock screen widgets
- 4K resolution support
- Restricted profiles for tablets
- Dial Pad auto-complete
- Shows the percentage of download and the time remaining.

Although, the Codename was changed from Ice Cream Sandwich to Jelly Bean but the numeric series had been kept the same as the previous one. And to be practical there was not much of difference in the feel of this version.

Android Version 4.4.x – Kitkat (31st October, 2013)

This was the most controversial Android Version of all time. The ever awaited LG Google Nexus 5 and the Android Version, Codenamed Kitkat, were to make their way into the spotlight together. Google had gone way too late on their release of this version and their smartphone. Although, both of them things were worth waiting for. The features included with in this version of Android were as follows:

- Screen Recording
- Translucent System UI
- Better and Enhanced notification access
- Performance improvements

Although, the list of changes might seem shorter than expected but Google actually made it look like something huge and they did succeed in the process. The contradictory part is also true, since a lot of people choose other devices over the Nexus because they arrived earlier and they seemed to be a better buy.

Android Version 5.0 – Lollipop (17th October, 2014)

With the release of this Android version it seemed like Kitkat (The previous Android Version) had not been in the market for long enough to make its way towards the hearts of the Android users. This version came with the Motorola Nexus 6 from the LG Nexus, which actually made it seem like they must have made more changes and after using this Android Version you would actually feel like that is true but that would only be true when it comes to the UI and the keyboard features.

Although this Android Version is still in its immature stage but the features they have come up with till date are as follows:

- New Design – Material Design
- Speed Improvement
- Battery Efficiency

Case Study on Android

Android 6.0 – Marshmallow May 28, 2015

- Android 6.0 "**Marshmallow**" was unveiled under the codename "Android M" during Google I/O on May 28, 2015, for the Nexus 5 and Nexus 6 phones, Nexus 9 tablet, and Nexus Player set-top box, under the build number MPZ44Q.
- Requirements for the minimum amount of RAM for devices running Android 5.1 range from 512 MB of RAM for normal-density screens, to about 1.8 GB for high-density screens. The recommendation for Android 4.4 is to have at least 512 MB of RAM,[213] while for "low RAM" devices 340 MB is the required minimum amount that does not include memory dedicated to various hardware components such as the baseband processor.

- The **Android N** Developer Preview is already here and this will be followed by monthly updates until the final version. That final version will likely come around Nexus time – late September or early October – with Android N availability for other manufacturers and devices in the six or so months to follow.

2. Explain design goals of Android.

A number of key design goals for the Android platform evolved during its development:

1. Provide a complete open-source platform for mobile devices. The open-source part of Android is a bottom-to-top operating system stack, including a variety of applications, that can ship as a complete
   product.

2. Strongly support proprietary third-party applications with a robust and stable API. As previously discussed, it is challenging to maintain a platform that is both truly open-source and also stable enough for proprietary third-party applications. Android uses a mix of technical solutions (specifying a very well-defined SDK and division between public APIs and internal implementation) and policy requirements
   (through the CDD) to address this.

3. Allow all third-party applications, including those from Google, to compete on a level playing field. The Android open source code is designed to be neutral as much as possible to the higher-level system
   features built on top of it, from access to cloud services (such as data sync or cloud-to-device messaging APIs), to libraries (such as Google's mapping library) and rich services like application stores.

4. Provide an application security model in which users do not have to deeply trust third-party applications. The operating system must protect the user from misbehavior of applications, not only buggy applications that can cause it to crash, but more subtle misuse of the device and the

user's data on it. The less users need to trust applications, the more freedom they hav e to try out and install them.

5. Support typical mobile user interaction: spending short amounts of time in many apps. The mobile experience tends to involve brief interactions with applications: glancing at new received email, receiving and sending an SMS message or IM, going to contacts to place a call, etc. The system needs to optimize for these cases with fast app launch and switch times; the goal for Android has generally been 200 msec to cold start a basic application up to the point of showing a full interactive UI.

6. Manage application processes for users, simplifying the user experience around applications so that users do not have to worry about closing applications when done with them. Mobile devices also tend to run without the swap space that allows operating systems to fail more gracefully when the current set of running applications requires more RAM than is physically available. To address both of these requirements, the system needs to take a more proactive stance about managing processes and deciding when they should be started and stopped.

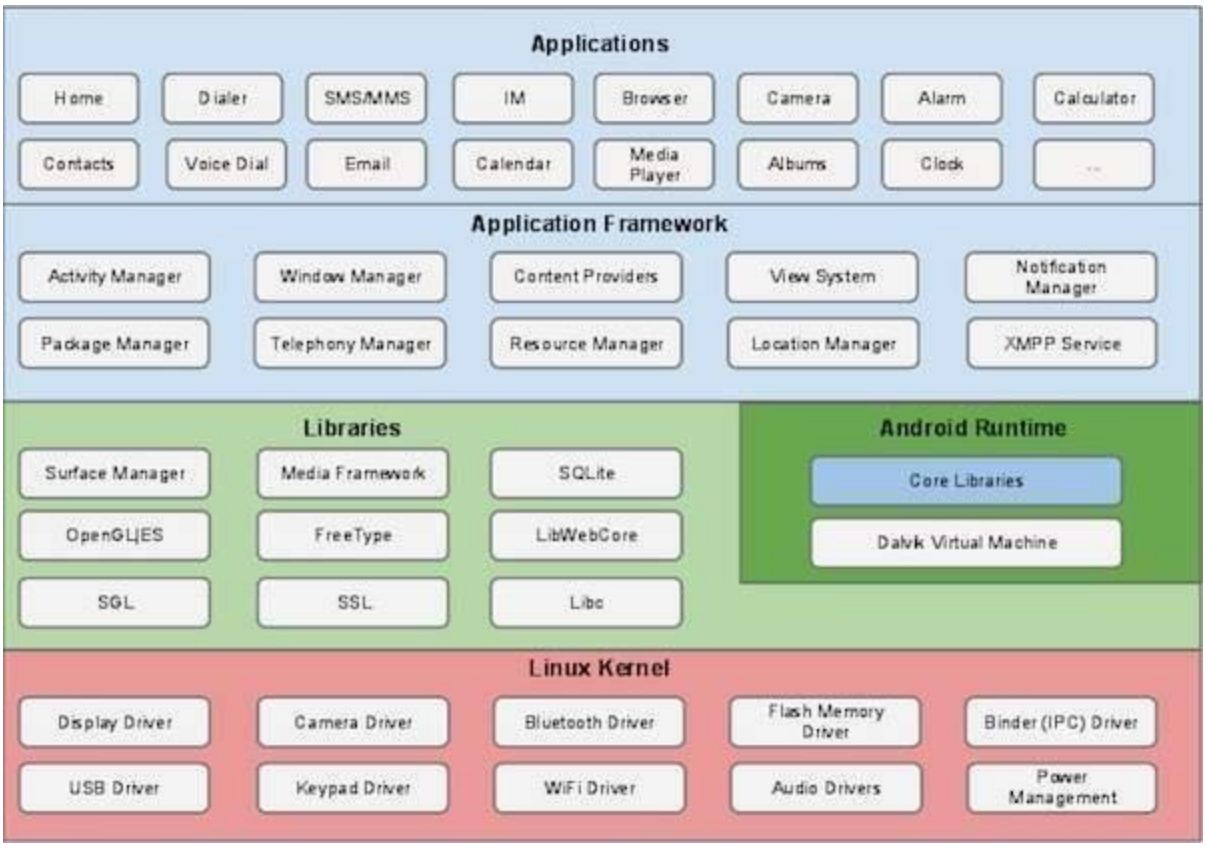
7. Encourage applications to interoperate and collaborate in rich and secure ways. Mobile applications are in some ways a return back to shell commands: rather than the increasingly large monolithic design of desktop applications, they are targeted and focused for specific needs. To help support this, the operating system should provide new types of facilities for these applications to collaborate together to create a larger whole.

8. Create a full general-purpose operating system. Mobile devices are a new expression of general purpose computing, not something simpler than our traditional desktop operating systems. Android's design should be rich enough that it can grow to be at least as capable as a traditional operating system.

3. Explain Android Architecture with diagram.

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.

Case Study on Android



Linux kernel

At the bottom of the layers is Linux - Linux 3.6 with approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential 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.

Libraries

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library 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.

Android Libraries

This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include 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.

Having covered the Java-based core libraries in the Android runtime, it is now time to turn our attention to the C/C++ based libraries contained in this layer of the Android software stack.

Android Runtime

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

The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. 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.

Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of 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.
- **View System** − An extensible set of views used to create application user interfaces.

Applications

You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games etc.

Android Compatibility

- Obtain the Android software source code. This is the source code for the Android platform that you port to your hardware.
- Comply with Android Compatibility Definition Document (CDD). The CDD enumerates the software and hardware requirements of a compatible Android device.
- Pass the Compatibility Test Suite (CTS). You can use the CTS (included in the Android source code) as an ongoing aid to compatibility during the development process.

4. Write a note on Binder IPC.

- Android provides the process-unit component model. In other words, system services which provide an application or a camera feature created in Java on Eclipse, or system services which are responsible for screen display, all such Android components are eventually expressed as Linux processes.

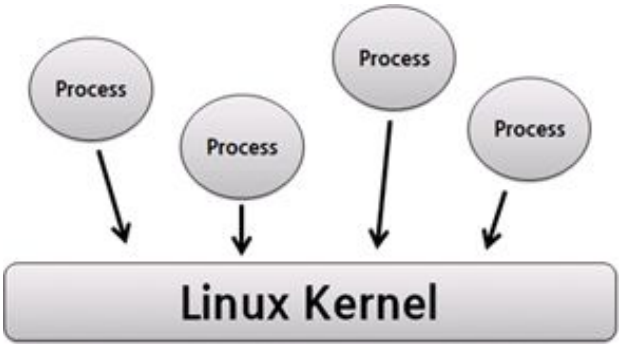


Figure: Process Management of Linux Kernel.

- Since Android runs based on Linux memory, process and file management, the system service is also isolated by Linux process for protection. The system process is written in Java code as well as C++ native code, so the services that run in Dalvik VM, such as Wi-Fi, Location and Activity Service, are included here.
- To support mobile devices such as smartphones, all of the default system functions of Android are provided as the *server process* type. In other words, to use the functions such as SurfaceFlinger or AudioFlinger, a request should be made as a separate process that runs on the user mode.

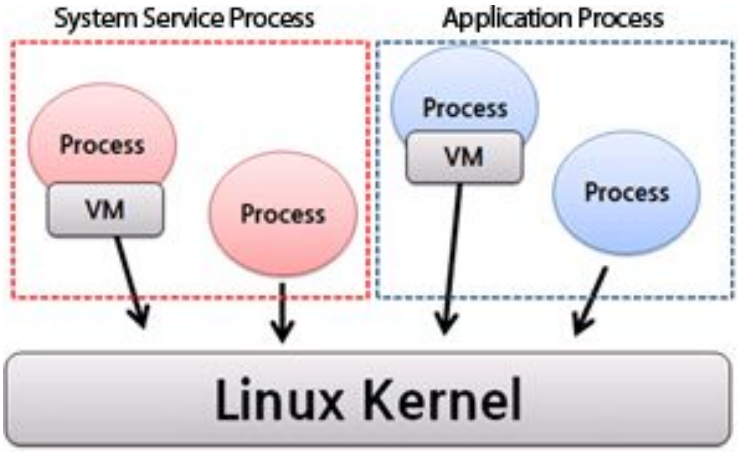


Figure: Android System Services and Applications Run by Linux Process.

- For example, when an application that have created calls APIs to get the location information provided by the Android SDK, the application sends a request to the Linux

process that provides the Location service internally and gets the response. Also, it interoperates with the camera service when a camera is used.

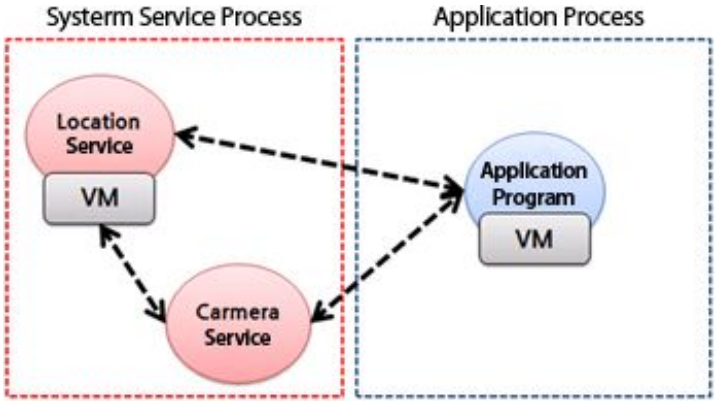


Figure: Application Process Which Calls Location and Camera Services.

- As all system services are provided as a *server process*, a mechanism to send requests and responses to other processes is necessary. In Android it is called the Binder (Context Switching). Android uses functions provided by other processes through Binder.

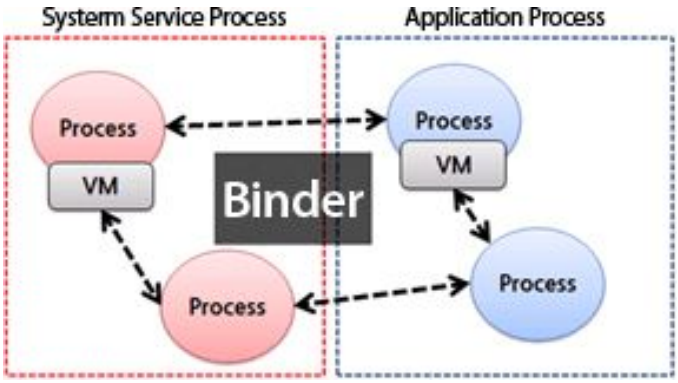


Figure: Binder, Android's Communication Infrastructure between Processes.

- It is because of performance. As we have discussed above, all system functions of Android are provided as a server process, so an optimized communication method between processes is required, and Binder is the result of this consideration. Binder refers to a kernel memory which is shared between all processes to minimize the overhead caused by memory copy. In addition, it provides the Remote Procedure Call (RPC) framework written in C++ for high productivity.

Case Study on Android

"*What are the benefits of the Android platform structure?*"

- The system functions are provided as server processes where requests and responses are received through the Binder mechanism.
- Easy to expand or remove functions: It is easy to add a new system service or remove an existing function.
- Easy to port: Porting to a new processor requires few changes. A toolchain for porting is provided.
- Easy to test: Tests are limited by the component unit, so it is not necessary to test the entire services, and more strict tests are available.
- Support for distribution system: Process communication is based on the Binder, so it guarantees transparency in location between components.

Binder that Binds All Functions

- The Binder mechanism has started from a simple idea. "*Let requests and responses be written in an area where all processes can share and let each process refer to the memory address.*" So, the kernel space is used for it.

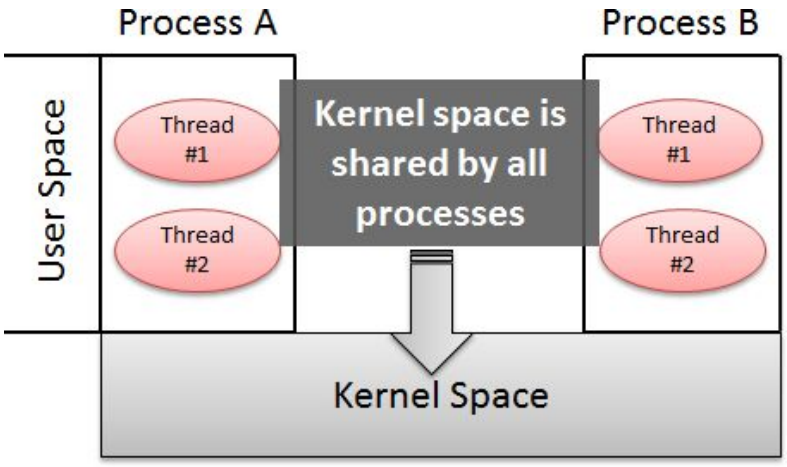


Figure: Kernel Space Shared by All Processes.

- A Binder Driver is implemented to use the kernel space. The role of the Binder driver is to convert the memory address that each process has mapped with the memory address of the kernel space for reference.

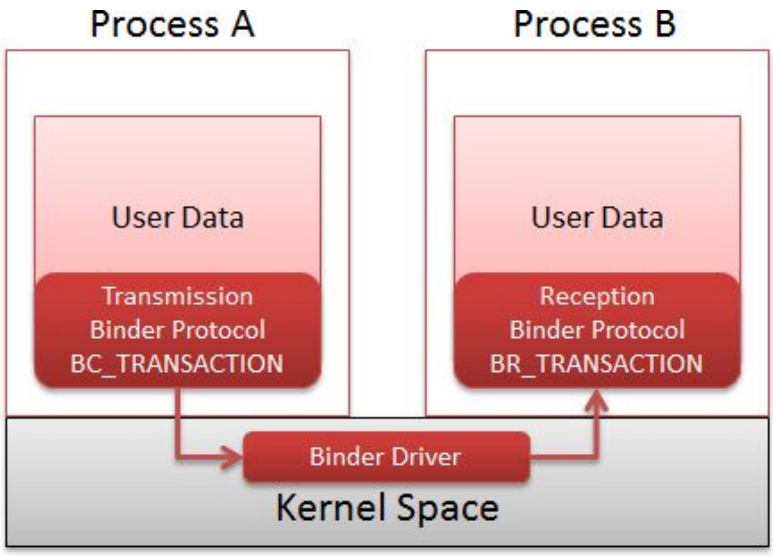


Figure: Binder Driver Configuration.

- The Binder driver can be used by the ioctl()(input output control)system function, which is a standard method in Linux. The mechanism is called Binder IPC.
- There is a C++ framework that processes the data transferred using the Binder IPC and makes Remote Procedure Call (RPC). It is used to create a system service. Then, a process can use functions provided by other processes as if their own.

Zygote

- Zygote is a daemon which only mission is to launch applications. This means that **Zygote is the parent of all App process**. When app_process launches Zygote, it creates the first Dalvik VM and calls Zygote's main () method.

5. Explain Dalvik Virtual Machine.

- As we know the modern JVM is high performance and provides excellent memory management. But it needs to be optimized for low-powered handheld devices as well.
- The **Dalvik Virtual Machine (DVM)** is an android virtual machine optimized for mobile devices. It optimizes the virtual machine for *memory*, *battery life* and *performance*.
- Dalvik is a name of a town in Iceland. The Dalvik VM was written by Dan Bornstein.
- The Dex compiler converts the class files into the .dex file that run on the Dalvik VM. Multiple class files are converted into one dex file.

- Java Virtual Machine: javac (Compiler)

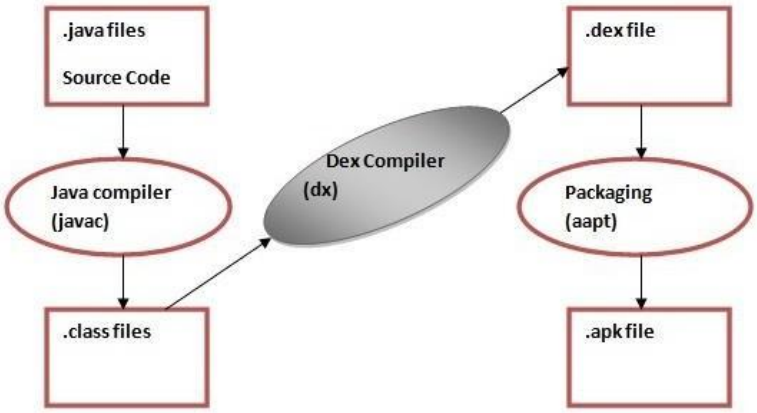
.java -> .class

Case Study on Android

- Dalvik Virtual Machine

.java -> .dex

Let's see the compiling and packaging process from the source file:



- The **javac tool** compiles the java source file into the class file.
- The **dx tool** takes all the class files of your application and generates a single .dex file. It is a platform-specific tool.
- The **Android Assets Packaging Tool (aapt)** handles the packaging process.

Wake Locks

- To avoid draining the battery, an Android device that is left idle quickly falls asleep. However, there are times when an application needs to wake up the screen or the CPU and keep it awake to complete some work.

1. **Keep the Screen On**
   - Certain apps need to keep the screen turned on, such as games or movie apps. The best way to do this is to use the FLAG_KEEP_SCREEN_ON in your activity. The advantage of this approach is that unlike wake locks (discussed in Keep the CPU On), it doesn't require special permission, and the platform correctly manages the user moving between applications, without your app needing to worry about releasing unused resources.

2. **Keep the CPU On**

   - If you need to keep the CPU running in order to complete some work before the device goes to sleep, you can use a PowerManager system service feature called wake locks. Wake locks allow your application to control the power state of the host device.
   - Creating and holding wake locks can have a dramatic impact on the host device's battery life. Thus you should use wake locks only when strictly necessary and hold them for as short a time as possible. For example, you should never need to use a wake lock in an

activity. As described above, if you want to keep the screen on in your activity, use FLAG_KEEP_SCREEN_ON.

- One legitimate case for using a wake lock might be a background service that needs to grab a wake lock to keep the CPU running to do work while the screen is off. Again, though, this practice should be minimized because of its impact on battery life.

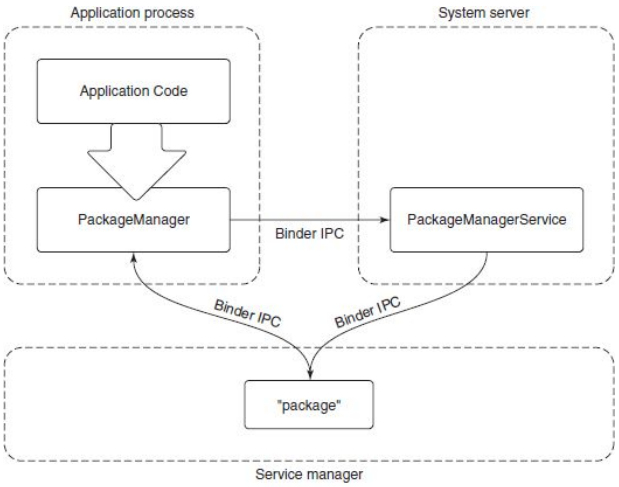


Figure 10-40. Publishing and interacting with system services.

Out-Of-Memory Killer

- Main difference is how LMK and OOM chooses a process to kill in low memory conditions.
- Main reason for introduction of LMK in android was OOM killer sometimes kill high priority process (Like foreground applications) in low memory conditions, on the other hand LMK has interface (oom score value) with activity manager (framework part) which knows priority of processes this results LMK always kill hidden and empty applications before killing foreground and active applications, apart from this some differences are below :

1) Low memory killer is utility designed for specially for Android for handling low memory condition whereas out of memory killer built in linux facility.

2)Low memory killer is invoked from kswapd process using shrink list where as oom killer is invoked using alloc_pages slowpath.

3)Low memory killer generally kill user space process, where as out of memory killer kill process randomly using some finest complex algorithm but never get killed itself.

4) Out of memory killer is last hope for kernel for freeing pages.

5) When out of memory killer is invoked it may result in kernel crash.

Case Study on Android

6. Give the Android Applications.

Open Handset Alliance

Open Handset Alliance is an amalgamation of Tech Companies with common and particular interest in the mobile user enhancement experience. Companies like Google, HTC, Motorola, Samsung, Telecom Italia, T Mobile, LG, Texas Instruments as well as Sony Ericsson, Vodafone, Toshiba and Hawaii are Tech giant based on their core abilities and strengths, while keeping and pursuing the characters and goals of each company, their basic idea of this joining of hands was the feature-rich mobile experience for the end user. This alliance meant the sharing of ideas and innovation, to bring out these ideas into reality. This provided the millions and millions of Mobile users the experience that they never had.

Like the Apple iphone, Android Operating System allows third party developers to innovate and create Applications and software for mobile devices. Android is an open, flexible and stable enough to associate itself with newer and newer evolving Technologies. Android's vast range of easy to use tools and wide range of libraries provides Mobile Application developers with the means of an amazing mobile operating software to come up with the most efficient and rich Mobile Applications changing the world of millions of mobile users.

Advantages & Disadvantages

Advantages:

Open- Android allows you to access core mobile device functionality through standard API calls.

All applications are equal- Android doesn't difference between the phone's basic & third party application even the dialer or home screen can be replaced.

Fast & easy development- The SDK contains what you need to build and run Android applications including different tools

Disadvantage:

Security- Making source code available to everyone indirectly invites the attention of black hat hackers.

Open Source- A disadvantage of open-source development is that anyone can scrutinize the source code to find vulnerabilities & write exploits
Login- Platform doesn't run on an encrypted file system & has vulnerable log-in

Case Study on Android

Applications

These are the basics of Android applications:

- Android applications are composed of one or more application components (activities, services, content providers, and broadcast receivers)
- Each component performs a different role in the overall application behavior, and each one can be activated individually (even by other applications)
- The manifest file must declare all components in the application and should also declare all application requirements, such as the minimum version of Android required and any hardware configurations required
- Non-code application resources (images, strings, layout files, etc.) should include alternatives for different device configurations (such as different strings for different languages)

Google, for software development and application development, had launched two competitions ADC1 and ADC2 for the most innovative applications for Android. It offered prizes of USD 10 million combined in ADC1 and 2. ADC1 was launched in January 2008 and ADC 2 was launched in May 2009. These competitions helped Google a lot in making Android better, more user friendly, advanced and interactive.

7. How Android supports security?

Google security services

Google provides a set of cloud-based services that are available to compatible Android devices with Google Mobile Services. While these services are not part of the Android Open Source Project, they are included on many Android devices. For more information on some of these services, see Android Security's 2015 Year in Review.

The primary Google security services are:

- Google Play: Google Play is a collection of services that allow users to discover, install, and purchase applications from their Android device or the web. Google Play makes it easy for developers to reach Android users and potential customers. Google Play also provides community review, application license verification, application security scanning, and other security services.

Case Study on Android

- Android updates: The Android update service delivers new capabilities and security updates to selected Android devices, including updates through the web or over the air (OTA).
- Application services: Frameworks that allow Android applications to use cloud capabilities such as (backing up) application data and settings and cloud-to-device messaging (C2DM) for push messaging.
- Verify Apps: Warn or automatically block the installation of harmful applications, and continually scan applications on the device, warning about or removing harmful apps.
- SafetyNet: A privacy preserving intrusion detection system to assist Google tracking and mitigating known security threats in addition to identifying new security threats.
- SafetyNet Attestation: Third-party API to determine whether the device is CTS compatible. Attestation can also assist identify the Android app communicating with the app server.
- Android Device Manager: A web app and Android app to locate lost or stolen device.

Security program overview

The key components of the Android Security Program include:

- **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 will be performed by external security experts upon release.
- **Open source and community review:** The Android Open Source Project 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 applications directly to users.
- **Incident Response:** Even with all of 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 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

(over-the-air updates), removing applications from Google Play, and removing applications from devices in the field.

- **Monthly security updates:** The Android security team provides monthly updates to Google Nexus devices and all of our device manufacturing partners.

Platform security architecture

Android seeks to be the most secure and usable operating system for mobile platforms by re-purposing traditional operating system security controls to:

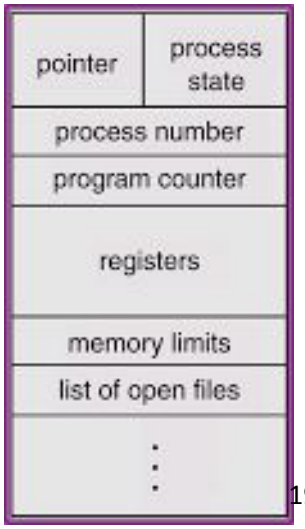
- Protect application and user data
- Protect system resources (including the network)
- Provide application isolation from the system, other applications, and from the user
- To achieve these objectives, Android provides these key security features:
- Robust security at the OS level through the Linux kernel
- Mandatory application sandbox for all applications
- Secure interprocess communication
- Application signing
- Application-defined and user-granted permissions

8. Explain process life cycle in Android.

In order to launch new processes, the activity manager must communicate with the *zygote*. When the activity manager first starts, it creates a dedicated socket with *zygote*, through which it sends a command when it needs to start a process. The command primarily describes the sandbox to be created: the UID that the new process should run as and any other security restrictions that will apply to it. *Zygote* thus must run as root: when it forks, it does the appropriate setup for the UID it will run as, finally dropping root privileges and changing the process to the desired UID.

Process Management Overview

- Process management in a typical operating system involves many complex data structures and algorithms, but doesn't go much beyond the level managing the typical process data structure. Android is similar in that at the base level the control structures look the same. Similar to this:



The details of each step in the illustration are:

1. Some existing process (such as the app launcher) calls in to the activity manager with an intent describing the new activity it would like to have started.

2. Activity manager asks the package manager to resolve the intent to an explicit component.

3. Activity manager determines that the application's process is not already running, and then asks *zygote* for a new process of the appropriate UID.

4. *Zygote* performs a fork, creating a new process that is a clone of itself, drops privileges and sets its UID appropriately for the application's sandbox, and finishes initialization of Dalvik in that process so that the Java runtime is fully executing. For example, it must start threads like the garbage collector after it forks.

5. The new process, now a clone of *zygote* with the Java environment fully up and running, calls back to the activity manager, asking ''What am I supposed to do?''

6. Activity manager returns back the full information about the application it is starting, such as where to find its code. New process loads the code for the application being run.

7. Activity manager sends to the new process any pending operations, in this case ''start activity X.''

8. New process receives the command to start an activity, instantiates the appropriate Java class, and executes it.

Category	Description
SYSTEM	The system and daemon processes
PERSISTENT	Always-running application processes
FOREGROUND	Currently interacting with user
VISIBLE	Visible to user
PERCEPTIBLE	Something the user is aware of
SERVICE	Running background services
HOME	The home/launcher process
CACHED	Processes not in use

Process importance categories

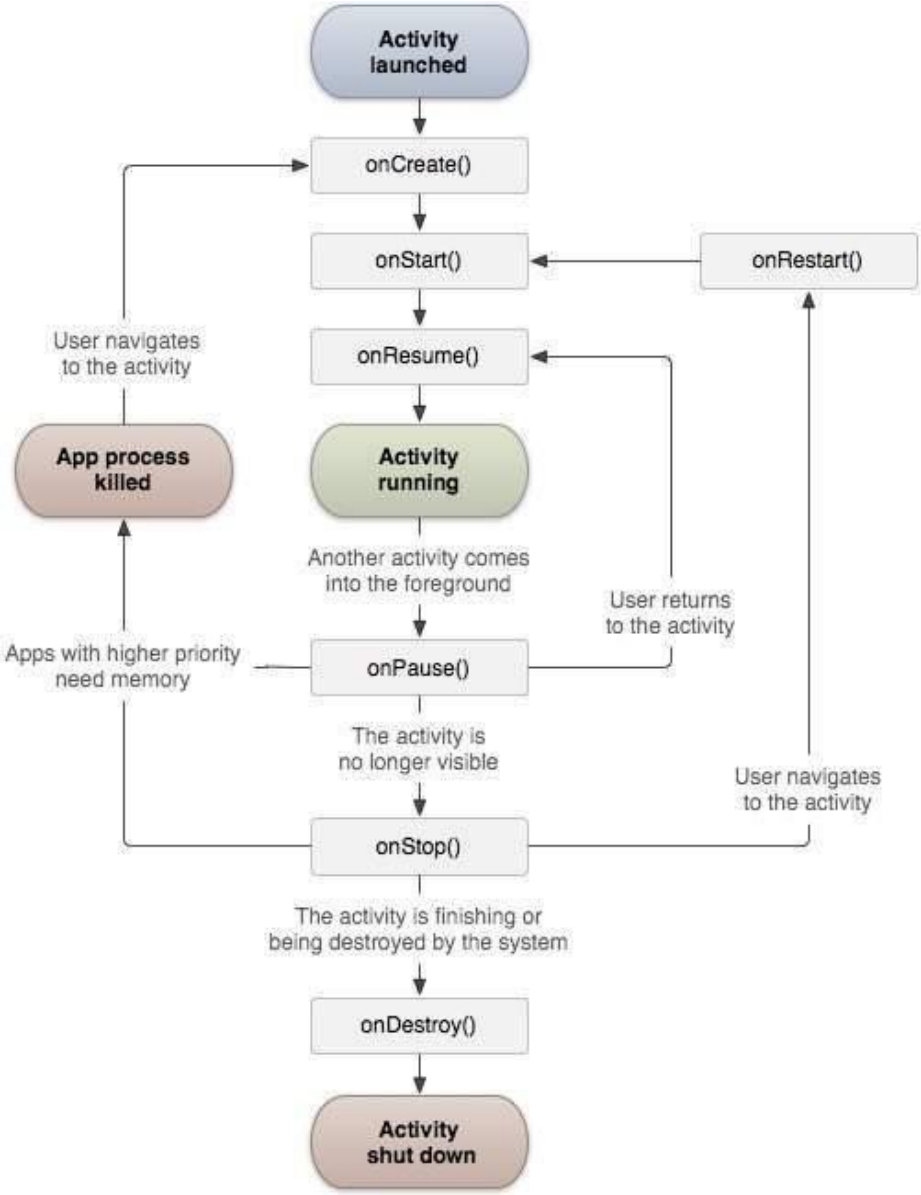
9. Explain any 5 life cycle events of android activities.

Process	State
system	Core part of operating system
phone	Always running for telephony stack
email	Current foreground application
camera	In use by email to load attachment
music	Running background service playing music
media	In use by music app for accessing user's music
download	Downloading a file for the user
launcher	App launcher not current in use
maps	Previously used mapping application

Process	State
system	Core part of operating system
phone	Always running for telephony stack
email	Current foreground application
music	Running background service playing music
media	In-use by music app for accessing user's music
download	Downloading a file for the user
launcher	App launcher not current in use
camera	Previously used by email
maps	Previously used mapping application

If you have worked with C, C++ or Java programming language then you must have seen that your program starts from **main()** function. Very similar way, Android system initiates its program with in an **Activity** starting with a call on*onCreate()* callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity life cycle diagram: (*image courtesy : android.com*)

Case Study on Android



The Activity class defines the following call backs i.e. events. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

Callback	Description
onCreate()	This is the first callback and called when the activity is first created.
onStart()	This callback is called when the activity becomes visible to the user.
onResume()	This is called when the user starts interacting with the application.
onPause()	The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
onStop()	This callback is called when the activity is no longer visible.
onDestroy()	This callback is called before the activity is destroyed by the system.
onRestart()	This callback is called when the activity restarts after stopping it.