Volume 4, Issue 1, January 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Android OS: A Robust, free, Open-Source System for Mobile Device

Sangita Dahiya[*]
M.tech (NS)&BPSM University
India

Kawarjeet Pahal
Lect in BITS College
India

Vinod Saroha
Lect. in BPSM University
India

Abstract: Mobile devices have seen an extensive amount of development in recent years, but one question is still looming and nobody seems to have the answer: what is 'standard' for the mobile platform? Many companies have already written their own in-house operating systems for the devices they manufacture such as Symbian or iPhone OS. However, with the existence of so many closed-source operating systems, no rational company would want to disclose their secrets and lose their edge on the competition. This presents a problem where software developers can't write their code to be generalized. The Android team hopes to solve this on two levels. Firstly, it seeks to arrive at a common open-source operating system that any mobile device can run on. Secondly, it seeks to make developing applications for these mobile phones more gene ral and hardware-agnostic. Android has become a very popular operating systems for smartphones and tablets but at the same time threats associated to this platform, like malware or exploits, are also growing. During this paper the author will describe how it is possible to improve the security of Android in order so it can be safely used in business. *Earlier this year 2014, the company's research team that focuses on solving this problem showed us Deep Short, a system for "capturing" an application like Google Maps that runs on a large screen via a smartphone camera and transfer it to mobile. Today, the team is turning this idea on its head and showing a cool new way to project smartphone content onto any display.*

Keywords:

I. Falling back on the traditional open-source method

COMMONLY, the problem that most any computing platform will have is interoperability, and one of the worst places to run into this problem is in a situation where the user has little to no control over what operating system they use. Eventually, software developers will label one platform obsolete and refuse to develop for it any more. So how can an expandable, compatible operating system that can be used on many architectures be created? Android OS attempts to solve this problem by building on existing technologies like Linux and applying them to the mobile device. Android OS is free to use, improvable, and designed with multiple hardware implementations. Best of all, it is open source, so any necessary changes that need to be made can be performed as low as the kernel level. Android OS is already catching the public eye since it carries the Google namesake, but still has yet to reach its full potential.

II. Humble beginnings

Four years ago, almost nobody had heard of Android, Inc., the small mobile software developer that Google acquired in July 2005. The acquirement shed little light as to exactly what Google was up to, only that Google was interested in the people working at Android [?]. The startup company consisted of a few notable people, one of them being Andy Rubin, who contributed in making the T-Mobile Sidekick. Rubin would soon become the leader of the Android project at Google. When news of Google's acquirement of a mobile software company hit the mainstream media, speculations were rampant; talk of a "GPhone" handset was on many Internet news outlets. In a few years the formation of the Open Handset Alliance removed some of the fog surrounding Google's actions, and showed that the "GPhone" was really just Google trying to enter the operating systems market for mobile devices [?]. Android OS became the first product created by the Open Handset Alliance, which consisted of many major technology vendors besides Google. Today, Android happily hums along as an open-source project. At the time of this writing, the HTC Dream is the only Android-enabled mobile phone on the market except for the mobile phone offered to Android developers.

III. Android OS architecture

Android can be subdivided into four main layers: the kernel, libraries, applications framework,and applications. As previously mentioned the kernel is Linux. The libraries that come with Android provide much of the graphics, data storage, and media capabilities. Embedded within the libraries layer is the Android runtime which contains the Dalvik virtual machine, which powers the applications. The applications framework is the API that all applications will use to access the lowest level of the architecture [?].

A. The kernel layer

As previously mentioned the kernel layer is Linux. Linux was chosen since it has a proven track record in desktop systems and in many cases doesn't require drivers to be rewritten. Linux provides such things as virtual memory, networking, drivers, and power management. Upon examining the kernel shipped with the Android source code, there are not any significant changes to the core functions of the kernel.

B. Native libraries layer

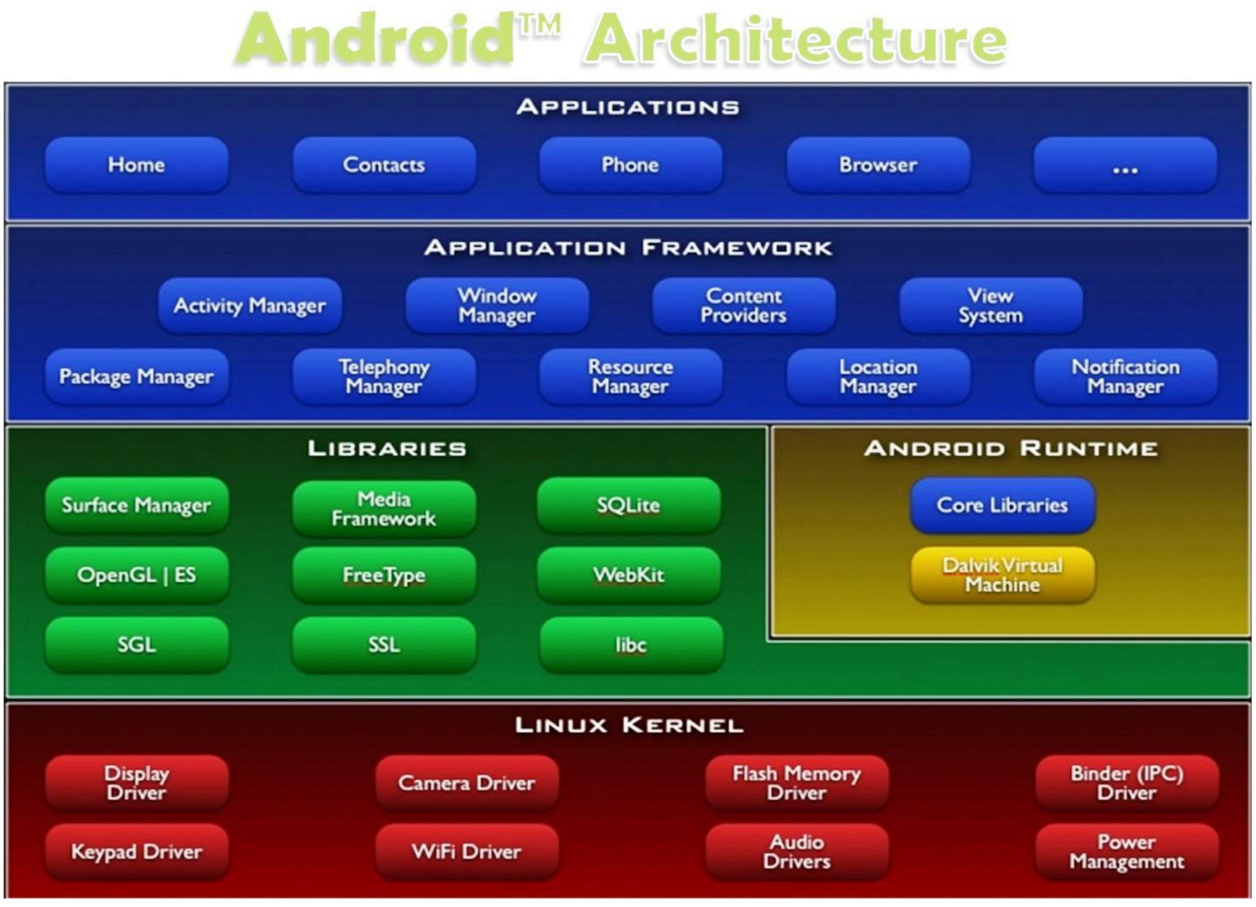
The native libraries layer provides Android with the capabilities for its core features. Android is shipped with SGL which acts as the primary 2D graphics renderer. Its counterpart is OpenGL ES which provides 3D graphics support. Android comes packaged with SQLite which takes care of most data storage. The WebKit web rendering engine is also shipped with Android and has been tailored to render web pages for smaller screen sizes. Of particular interest is the Dalvik virtual machine which is a part of this layer. The Dalvik virtual machine is a bytecode interpreter which is highly optimized for executing on the mobile platform. The bytecodes are converted Java binaries that are very quick and efficient to run on smaller processors. The core libraries are written in Java and provide much of the core classes which would normally be available in a Java virtual machine.

C. Applications framework layer

This layer and the layer above it are written completely in Java. The applications framework provides all of the major APIs that the applications will use including things like sharing data, accessing the telephony system, and receiving notifications. An important thing to note about Android OS is that all applications use this same framework no matter the author of the application. This is quite a departure from what many other mobile OS designers have chosen to do. For instance the iPhone most certainly differentiates between Apple software and third-party software down to the copy-and-paste feature.

D. Applications layer

All of Android's software is written in Java, which is interpreted by the Dalvik virtual machine.



Even the most core features such as the phone and the contacts application reside in this layer. This layer contains software written by the Android team as well as any third-party software that is installed on the device. An effect of allowing third-party developers access to this layer is that the user interface can be overhauled comparatively easily. Third party applications can handle any event that the Android team's application could see (such as the phone ringing). This means that so long as there is a replacement application for the dialer application, anyone could potentially write their own. Given this model we might expect that, as Android becomes more robust, the user will be able to specify what applications should handle which events.

IV. Android system services

Android provides the services expected in a modern operating system such as virtual memory, multiprogramming, and threads, all on a mobile platform. Many of Android's services are a result of including the Linux kernel. However the Android team has added the telephony stack in return.

A. The Linux CPU scheduling algorithm

Linux employs a number of different methods for scheduling its processes and its algorithm is very nontraditional in the big picture. There are three scheduling schemes for Linux. Each process is assigned a scheduling scheme depending on the type of task it presents. Real time tasks will often run in the SCHED FIFO or SCHED RR scheme. All other tasks will run in SCHED NORMAL [?]. SCHED NORMAL tasks are handled by a special algorithm and are preempted by SCHED RR and SCHED FIFO tasks. In a few words the algorithm behaves like a hybrid priority queue that rewards process which are not CPU-greedy. CPU time is divided up into epochs, which are equal slices of time in which the processes can run and use up some of their allotted time, or timeslice. At the end of every epoch, each process' remaining timeslice is halved. More time is added proportional to the processes' nice value, a value specified by the user or by the system's default nice value. The interesting rollover time has a rewarding effect to I/O bound processes. Since I/O bound processes will likely spend a lot of time waiting, the scheduling algorithm rewards it by letting it keep some of its timeslice. This way processes which are likely being used by the user are very responsive and will often preempt more non-interactive, CPU-bound processes like batch commands. SCHED RR tasks will preempt SCHED NORMAL tasks and are preempted by SCHED FIFO tasks. These tasks are scheduled according to round-robin rules within their own priority bracket. SCHED FIFO tasks behave quite like the name would suggest. Tasks in this queue are scheduled by priority and arrival time, will preempt any other processes trying to run, and will execute for as long as they please.

B. Android and file systems

Android makes use of a multiplicity of file system types, namely due to the expectation of external memory with unsure file system types. As a result Android relies on the standard Linux package to provide for file systems such as ext2 and ext3, vfat, and ntfs. Android itself uses the yaffs2 file system, which is not a part of the standard Linux kernel, as its primary file system. YAFFS is a file system optimized for NAND and NOR flash memory. At the time when it was developed, file systems did exist for flash memory but most of them catered toward chips which were small enough to use small block sizes. This was unsuitable for large NAND flash chips. YAFFS attempts to solve this by abstracting storage to ”chunks” which scale according to the page size. To scale the method up for considerably large NAND devices, YAFFS has a tweak in the way that it addresses pages. For instance, we might use a page address size of 216 and we may have have 218 chunks to address. We do not have sufficient capabilities to address pages individually, but we can address groups of four pages and search for the desired page from there. The useful thing about this is that now we have the option of neglecting to use RAM at all to augment our file system. We can pretty easily used some sort of indexed referencing scheme to build a file from a string of chunk IDs, and any scanning for particular pages within that chunk will be a limited set of a size equal to the number of chunks divided by the address size. (218 ÷ 216 = 4). This linear probing may seem like a bad idea. In practice, the inefficiency is too small to notice on such a small set of chunks [?].

YAFFS is preferable as a file system in Android since it optimizes the use of NAND devices as storage and also has great efficiency in memory usage.

C. The radio interface layer

Consistent with the rest of the design of Android, the Android team has created a way to abstract a phone call. Since the way that handset manufacturers implement the radio device on cellular devices will inevitably vary, Android has to remain ignorant to the way software interacts with hardware to place the call. The Android team solves this problem by creating the Radio Interface Layer Daemon (RILD), which is the way that the applications framework interfaces with the shared libraries. The RILD's main function is to provide event-driven middle ground between the applications framework and the radio drivers. The RILD is part of the Radio Interface Layer (RIL), which consists of two major parts: the Android RIL and the Vendor RIL. The Android RIL includes the applications framework which makes the request to the RILD to place a phone call, while the Vendor RIL is the part that is up to the vendor to implement [?]. The Vendor RIL includes the driver that the vendor ships for the hardware implementation of the radio antenna. This model allows any hardware to be implemented for placing phone calls so long as the vendor writes drivers which follow the model.

V. Similarities to iPhone OS

In many ways, it seems as if Android was designed to compete directly with the iPhone OS. The Android team saw that iPhone OS presents an intimidating level of excellence in user interfacing that Android would inevitably have to compete with.

A. The touchscreen

Android has the full capabilities of interfacing with a touchscreen-enabled device, and Android is multitouch-capable[?] but is unimplemented. Many believe this is for Google to remain in good stead with Apple. In any case, the Android implementation on the HTC Dream does show some similarity to the iPhone in the touch-gesture design, such as flicking your finger across the screen to flip to the next page of items. A notable gesture that Android lacks compared to iPhone OS is the zooming gesture, where two fingers are touched on the screen and moved away or toward each other to zoom in and out of the image. In the iPhone implementation of Google Maps, this gesture is available, but in the Android version, ”zoom in” and ”zoom out” buttons take its place.

B. Modular application design

The iPhone is heralded for its multitude of stand-alone ”apps” that are available. Android seems to have mirrored this model exactly. However Android differs in its underlying design for Android offers third-party applications unbridled access to the applications framework layer. Modular applications tend to ease the process of understanding to the user and help the user visualize the application as a single, contained unit rather than a string of dependent software.

C. The Android Market vs. The App Store

Similarly both operating systems offer a way for developers to publish their applications either for free or for a price. Also users can visit and download applications from the store. Many of the popular apps on the iPhone store are beginning to pop up on the Android Market as well implemented for placing phone calls so long as the vendor writes drivers which follow the model.

VI. Security Threats in Android

Android is an Operating System based on LINUX so we can say that it has the same functionalities as any desktop running a modern OS with Internet access, but with some additional hardware such as camera, GPS, etc.This means that the same threats that apply to modern OS can be extrapolated to Android, like malware or exploits.However, in the case of smartphones, there is an important difference which can have a high impact: mobility. Mobility means that a static network is not assigned to the device, as in the case of desktops or servers, hence connecting to any networks through 3G/UMTS orWiFi
exposes the device to any kind of network attack (sniffing,spoofing, etc). Because the device is not assigned to any fixed network this gives the user some flexibility to move/travel, but it also results in a lack of some security layers of protection such as external firewalls,perimetrical antivirus or IDS/IPS which are found in any business. The article by Bastian Knings, Jens Nickels, and Florian Schaub , describes a good example of how it is possible to exploit a vulnerability in Android through a WiFi connection. They explain how an attacker can eavesdrop and access the Google Calendar content and even impersonate the user. Another good example of a tool which takes advantage of LAN/WiFi attacks is Firesheep , which can also be run in Android. Applications in Android can be installed in different ways. The most popular one is the official Google repository, Market. It is also possible to install packages from the shell connected to the USB. Any developer can deploy an application, even malware and distributed via the repository, as has happened in the past. Some Spanish security researchers introduced malware in Android as PoC in order to demonstrate the lack of security in the Market . In my opinion, this is the most critical part in the security chain. The end user must never be able to install any application and only authorized software should run in the device. Symantec's report 'The Current State of Mobile Device Security' highlights the same issue .Camera and GPS are not insecure by default, unless
the software running or drivers are vulnerable. The biggest issue with them is the lack of privacy, since GPS can be used to track the device and the photos taken with the camera can be stolen. Bluetooth is a different story because it allows the user to send and receive traffic, hence it can be used to control the device, steal confidential data, etc. Bluetooth is also useful to connect external devices like headphones or to connect to the car handsfree, however the author will not consider those scenarios in this project and bluetooth will be disabled. Another point to address is the physical security of the device. The same controls applied to laptops should be applied to smartphones: encryption, passwords policies to login, etc, and if possible remote wiping and remote GPS localization. The main problem is that versions of Android below 3.0 do not support encryption by default so it is not possible to encrypt the device itself. We should take into consideration if the SD card attached to the device is secure or not and the risk associated with it.The latest point to address is how to manage and administer the smartphone from a centralized location.
For instance, this will permit the security administrator of the business to install authorized software on the smart phone. By default, Android can be accessed from the shell with the SDK toolkit [8], but this gives the user the possibility of also accessing the device through USB, and this is a security breach of the policies.

VI1. Pushing for standardization

Android was designed with openness in mind. It would become the first widespread generalpurpose open operating system designed specifically for the mobile device. In the spirit of keeping the entire project open, the Android team has a few design goals to help support this ideal.

A. Design goals

The Android team often lists these four main points[?] as the main ideas of Android:
Open: Android tries to be especially developer-friendly, thereby making the device completely open for their applications to utilize. As well, Android is open-source and open to implement on most any hardware.
All applications are created equal: Android's core applications and third-party applications are on equal playing ground. They share the same API, have the same levels of access, and can do anything that the other does. Any core application can be replaced by any third-party application.
Breaking down applications boundaries: Android does not try to hide any functionality from the developer. The developer should easily be able to tap into the telephony system, GPS system, or essentially any thing that has been implemented in the applications framework.
Fast and easy application development: The applications framework makes it very easy to do some fairly sophisticated tricks like reporting the phone's current location and much more. This speeds up the development process for the developer and makes it easier to focus on design rather than implementation details.

B. Target hardware

Android makes no attempt at being specific to any one hardware and ships with drivers for many of the common types of hardware seen on the mobile phone. Android is not specific to any one type of mobile device but it is specific to the mobile device in general. However it is most likely that the next line of Android phones will be produced by members of the Open Handset Alliance and that Android will be well-suited to run on these platforms. The operating system itself does not have any preferences.

VII. An open business model

Android is fronted by a coalition of 47 technology and mobile companies, the Open Handset Alliance, many of them with extremely high revenue as it is. Money is not the the primary interest in the development of Android as it is free and open-source. However, Android presents a mutual interest between all these companies to create a mobile operating system that is extremely viable and widespread so that it becomes easy to concentrate a business model around some sort of standard. Since Windows, Mac OSX, and Linux are the biggest operating systems in the home computer and server market, they have well-cultivated developer bases. The Android team hopes Android will take off in the same way. It would be beneficial for a company to focus on developing software for one operating system that nearly everyone uses rather than several operating systems which have equal parts in user base. There is room for profit in the Android Market. Currently, the Android Market charges 25 USD to be a registered developer. As well, a developer can purchase and Android Developer's Phone for almost 400 USD. Royalties could potentially be charged for non-free applications on the Android Market as well.

References

[1] Ben Elgin, "Google buys android for its mobile arsenal," http://www.businessweek.com/ technology/content/aug2005/tc20050817_0949_tc024.htm, August 2005.

[2] John Cox, "Why google's gphone won't kill apple's iphone," http://www.networkworld.com/ news/2007/100807-google-gphone-iphone.html, October 2007.

[3] Mike Cleron, "Androidology: Architecture overview," http://www.youtube.com/watch?v= Mm6Ju0xhUW8, November 2007.

[4] Josh Aas, "Understanding the linux 2.6.8.1 cpu scheduler," Retrieved from http://www. angelfire.com/folk/citeseer/linux_scheduler.pdf, February 2005.

[5] Charles Manning and Wookey, *YAFFS Specifications*, Aleph One, Bottisham, UK, version 0.3 edition, February 2002, Retrieved from http://www.yaffs.net/yaffs-spec.

[6] Google, Mountain View, CA, *Radio Layer Interface*, March 2009, See path '/development/ pdk/docs/telephony.html' in Android source tree.

[7] Ryan Gardner, "Multi-touch proof-of-concept," http://ryebrye.com/files/multitouch_ archive.zip, November 2008, With instructions at http://www.ryebrye.com/blog/2008/ 11/30/g1-multitouch-proof-of-concept-soure-code-posted/.

[8] Open Handset Alliance, "Android overview," http://www.openhandsetalliance.com/ android_overview.html.

[9] www.google.com(latest research news)