Operating Systems for Mobile Devices

23/11/2012

1 Introduction

Small devices place many constraints upon the operating systems designed for them. This report aims to identify these constraints and explain what effects they have on the design of operating systems, with reference to the Android operating system in particular.

2 Body

2.1 Physical Space

Mobile devices, by their very nature, must be small. This places limitations on hardware; there is much less space for hardware in a mobile device than in a larger PC or Laptop. Necessarily, then, mobile devices will have more modest hardware specifications when compared to larger devices. This will in turn influence the design of any mobile OS, as it will need to run well and reliably with these more modest hardware capabilities.

2.2 Usage and Functionality

Mobile devices are used very differently from other devices, having fundamentally different requirements. A mobile phone, for example, must be constantly connected to some mobile network, and able to transmit/receive calls/texts at all times. This is the basic core functionality expected of a phone and as such needs to be considered in the design of any mobile OS, whereas an OS for a PC need not consider this as the functionality would not be required. Additionally, mobile devices utilize very different I/O devices; any modern mobile OS must be designed to handle a touchscreen/keypad, a camera, and have Bluetooth, GPS and WiFi capabilities, but probably does not need the capabilities to interface with peripheral devices (such as a printer), unless it be via Bluetooth or through a larger machine.

This is part of the reason why Android arrived at the following anatomy:

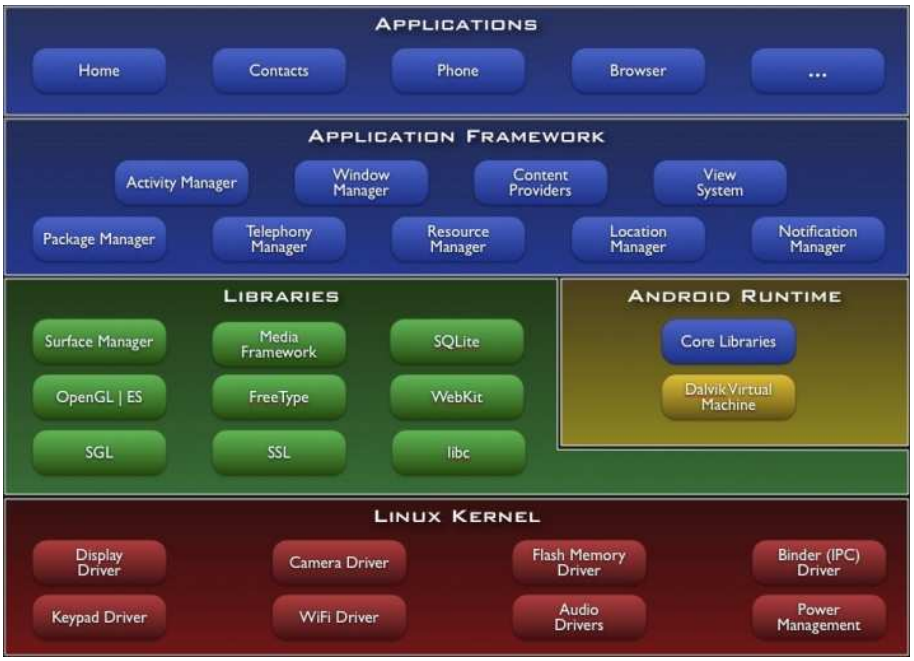

Figure 1: Android Anatomy
Source: http://developer.android.com/images/system-architecture.jpg

The Android anatomy is based on stripped down version of the Linux Kernel. It lacks much of the functionality of a Linux distribution; it does not have many standard Linux utilities, a native windowing system or Glibc[1], because this functionality is not necessary for a mobile device, and by omitting this functionality memory is saved. It makes use of the Dalvik Virtual Machine, chosen over JVM for its ability to run multiple virtual machines efficiently and its minimal memory footprint[2] (also possibly to get around licensing issues[3]). This will be discussed futher on in the report.

2.3 Batteries

Small devices such as Smartphones and Tablets typically run on battery power, and due to the constraints on physical space these batteries don't tend to be large, especially in mobile phones. However the device must still have a reasonably long battery life, (as long as reasonably possible), and as such the OS should aim to use power as economically as possible. To this end, the OS should keep the use of all hardware to a minimum. This influences not only the design of

the OS, but also the design of the device at a hardware level. Mobile devices tend to use chips made by ARM (with ARM holding roughly 95% of the market share in the smartphone market during 2010 [4]), whereas traditionally PCs have mainly used Intel x86 processors [5][6]. ARM chips currently use less power than ones made by Intel, making them more suited for use in battery operated devices[5], however PCs are generally designed without worrying quite so much about power expenditure, (since they generally have access to mains power), and so can afford to utilize x86 chips. As such, operating systems for mobile devices should aim to support the ARM architecture as much as possible.

In actuality, due to its being based on the Linux Kernel, Android has support for both the x86 and ARM architectures, as well as MIPS[7]. However most android devices are ARM based, making this rather redundant. Other mobile operating systems, such as iOS, Symbian and Windows Phone etc. are exclusively ARM[7].

With power conservation being such a big factor, mobile operating systems have to consider providing utilities to help preserve power as much as possible at both the OS and application level. None-mobile operating systems do this as well to a certain extent - to enhance battery life for Laptop Computers, but not to the same degree.

2.4 Hardware

The processors in small devices run at much lower speeds than modern day PCs and Laptops, and there tends to be significantly less Primary Memory. In 2011 smartphone processor speeds ranged from 1-1.6 GHz, and RAM ranged from 512mB - 2GB[9]. This is limiting for the OS, as the slower processor speeds and lower primary storage capacity affect the responsiveness of the device to the user, as well as the complexity of operations that can be expected to be reasonably performed. Additionally, mobile devices often have small amounts of secondary storage. This places limits on the amount of memory which can be used as virtual memory, which in turn impacts the device's ability to effectively multi-task. Consequently, the OS must be written in a way that ensures most economic processor and memory usage. Android addresses this in several ways.

Firstly, the Dalvik Virtual Machine utilizes a JIT (just in time) compiler (added in the Android 2.2 release)[8], to speed up the execution of applications (all applications are written in Java). According to Android developer Dan Bornstein, 'The JIT is a software component which takes application code, analyzes it, and actively translates it into a form that runs faster, doing so while the application continues

to run.' He adds that code can be observed running effectively 'about 4x to 10x faster than a more traditional interpreter implementation'. This is a very significant saving of CPU time, and whilst there is a tradeoff with memory usage Dan writes that 'The code for the JIT itself is well under 100k, and each process that the JIT runs in will typically only use another 100k or so of RAM'. Considering the saving the JIT can offer in terms of processing speed, it certainly seems worth the small tradeoff in memory.

Secondly, Android uses a process called Zygote when launching applications[1]. Zygote is a core process which launches at startup. It is initialized and core libraries are linked in. Whenever a new process is launched Zygote is forked, with the new process taking one of the forks. This results in 2 effects: each new process runs in its own virtual machine, protecting application files and memory without using additional system resources; and core libraries aren't copied and are stored in one place, saving memory. They will be copied should the application attempt to modify them, however.

Android also has its own method for terminating processes[1][10][11]. It is called lowmemkiller and starts when all caches have been emptied[11]. It terminates processes if and when the memory available to the OS drops below a certain threshold[11]. The order in which processes are killed is determined by their priority[10], which is determined by a least recently used algorithm[1]. From some of the comments of developer Arve Hjønnevåg in [11], it looks like this may have been a bit of a hack (which also makes it really difficult to understand). Nevertheless it solves memory issues - without it there have been issues of devices being 'stuck for many minutes before the system managed to allocate enough memory for the oom killer to kick in' [11].

One big drawback for Android, and Mobile operating systems in general, is that they can't really implement virtual memory due to lack of secondary storage. In the case of Android some apps do store their state when killed, but not all apps do and some states do not contain enough data. An example of this from [12], the browser stores URLs of open pages when killed, but not the data on them, meaning that the device has to reacquire the data from the internet, which is slow. This is probably rendered necessary by lack of secondary storage. Since it is built on the Linux Kernel, the capability of implement a swap partition is there and has been done [12], but it appears to be subject to debate as to whether this actually improves performance or not.

Another way in which Android saves memory again goes back to

the Dalvik Virtual Machine. A tool dx is used to convert some Java .class files into a .dex file, which typically is a few percent smaller than .jar file compiled from the same .class files [13]. There is also the possibility of a little further optimisation of the .dex when it is installed onto a mobile device. There is ongoing debate as to the merits Dalvik's register based approach versus the stack machines approach of Sun's JVM. Some sources suggest that a register based approach is faster [13], whereas others appear to suggest the opposite [14].

3 Conclusion

To summarise, I have identified several constraints on OS design for small devices: The small amount of physical space; The modest hardware, specifically the processor, RAM and secondary storage; and the different requirements with regards to functionality. Android has tried to tackle these constraints in various ways; with everything coming together to try and minimise CPU and Memory usage. It's use of the Dalvik virtual machine along with Zygote gives a level of abstraction between the Linux kernel and Java applications, ensures protection of applications' files and memory, reduces memory usage by storing libraries in one place, all without compromising CPU speed, and possibly improving it. It also attempts to optimise memory usage by killing background processes, with some level of success, and makes an attempt to store killed process states to help relaunch them faster. All these optimisations, along with Android's adoption of the ARM architecture, work together to help ensure better power conservation and hence longer battery life, improving the utility of the device.

4 Bibliography

[1] Colt. "Android OS - Processes and the Zygote!"
http://coltf.blogspot.co.uk/p/android-os-processes-and-zygote.html

[2] Williams, Alun; (2011/10/25). "What is... the Dalvik virtual machine?"
http://www.electronicsweekly.com/blogs/eyes-on-android-updates/2011/10/what-is-the-dalvik-virtual-machine.html

[3] Mazzocchi, Stefano; (2007/11/12). "Dalvik: how Google routed around Sun's IP-based licensing restrictions on Java ME"
http://www.betaversion.org/~stefano/linotype/news/110/

[4] Morgan, Timothy Prickett; (2011/02/01). "ARM Holdings"
http://www.theregister.co.uk/2011/02/01/arm_holdings_q4_2010_numbers/

[5] Bajarin, Tim; (2012/07/16). “ARM vs. Intel: How the Processor Wars Will Benefit Consumers”
http://techland.time.com/2012/07/16/arm-vs-intel-how-the-processor-wars-will-benefit-consumers-most/

[6] Courtland, Rachel; (May 2012). “The Battle Between ARM and Intel Gets Real”
http://spectrum.ieee.org/semiconductors/processors/the-battle-between-arm-and-intel-gets-real

[7] “Comparison of mobile Operating Systems”
http://en.wikipedia.org/wiki/Comparison_of_mobile_operating_systems

[8] Bornstein, Dan; (2012/05/25). “Dalvik JIT”
http://android-developers.blogspot.co.uk/2010/05/dalvik-jit.html

[9] “Comparrison of Smartphones”
http://en.wikipedia.org/wiki/Comparison_of_smartphones

[10] Unknown Author (2010/07/05). “Memory Management in Android”
http://mobworld.wordpress.com/2010/07/05/memory-management-in-android/

[11] Hjønnevåg, Arve et al. First Post (2009/01/13). “lowmemory android driver not needed?”
http://www.gossamer-threads.com/lists/linux/kernel/1020983

[12] Blattman, Jeffrey; Bob; (2009/12/18). “Why Android Swap Doesnt Make Sense”
http://zerocredibility.wordpress.com/2009/08/24/why-android-swap-doesnt-make-sense/

[13] Shi, Yunhe; Gregg, David; Beatty, Andrew; Ertl, M. Anton (2005-06-11). "Virtual Machine Showdown: Stack Versus Registers".
http://www.usenix.org/events/vee05/full_papers/p153-yunhe.pdf

[14] Vandette, Bob (2010-11-22). "Java SE Embedded Performance Versus Android 2.2". Oracle Corporation.
http://blogs.oracle.com/javaseembedded/entry/how_does_android_22s_performance_stack_up_against_ja