A Survey on Android vs. Linux

Frank Maker[1] and Yu-Hsuan Chan[2]
Department of Electrical and Computer Engineering, University of California, Davis[1]
Department of Computer Science, University of California, Davis[2]
{flmaker, yhcchan}@ucdavis.edu

1 INTRODUCTION

Android is an open source mobile device operating system developed by Google based on the Linux 2.6 kernel. The Linux kernel was chosen due to its proven driver model, existing drivers, memory and process management, networking support along with other core operating system services [1]. In addition to the Linux kernel various libraries were added to the platform in order to support higher functionality. Many of these libraries originate from open source projects; however the Android team created their own C library, for example, in order to resolve licensing conflicts. They also developed their own Java runtime engine, optimized for the limited resources available on a mobile platform called the "Dalvik Virtual Machine." Lastly, the application framework was created in order to provide the system libraries in a concise manner to the end-user applications. In this survey we highlight the major differences between a standard Linux operating system and Android. Particular emphasis is placed on the unique challenges present on the mobile embedded devices and how these differences motivate Android's structure with respect to Linux.

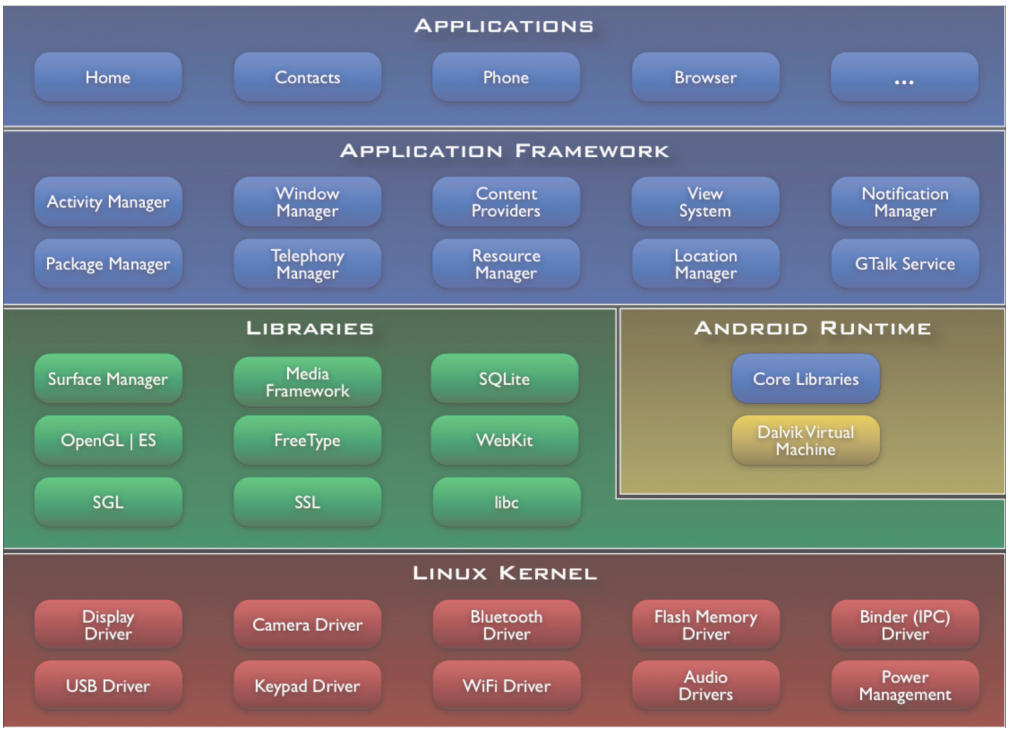

Figure 1 Architecture of Android [1]

2 TARGET ARCHITECTURE

The Linux kernel supports many different target architectures. However only two are fully supported by Android at this time: x86 and ARM. The x86 architecture for Android is mainly targeted mainly at Mobile Internet Devices (MIDs) whereas the ARM platform is prevalent on mobile phones. These architectures are typically used for very different computer systems. The x86 platform is used for general purpose desktop/laptop/server computing whereas ARM is widely in use on mobile devices. A comparison of these two platforms provides strong insight into the fundamental differences between Linux and Android.

Linux was initially created in 1991 as an open-source desktop operating system by Linus Torvalds. He created the Linux kernel since the MINIX operating system (an academic-oriented UNIX-like system) did not support the new 32-bit features of his Intel 80386 very well. The x86 family later came to dominate the desktop market and led to most personal computers using the x86 architecture today. Consequently much of the Linux kernel development today is focused on this family.

Smart phones in comparison do not have one clear market leader. Nokia, Research in Motion (RIM), Apple, HTC and Samsung all have a significant fraction of the smart phone market [2]. Despite no one manufacturer dominating the smartphone segment, they all primarily use one architecture, ARM. In fact as of 1996 98% of all mobile phones had at least one ARM microprocessor in them [3]. The widespread popularity of the ARM platform is largely due to its focus on power saving features. Unlike desktop systems, mobile phones rely on battery power for operation and therefore power consumption is a primary design factor.

Among the most fundamental differences between these two architectures is their instruction design philosophy. The x86 family is primarily a CISC (Complicated Instruction Set Computer) design whereas ARM is a RISC (Reduced Instruction Set Computer) architecture. This results in more, simpler instructions being used by ARM processors when compared to an equivalent set of x86 operations. Memory is at a premium in embedded devices due to size, cost and power constraints. ARM makes up for this concern by offering a second 16-bit instruction set called Thumb which can be interleaved with regular 32-bit ARM instructions. This additional instruction set can reduce code size by up to 30%, at the expense of some performance [4].

The ARM family also has a system-wide design focus on low power consumption. For example, the ARM7100 consumes only 72mW when operating at 14MIPS, 33mW in idle mode and 33uW during standby. In comparison the smallest of the Intel's x86 processors, the Atom, is designed for operation at roughly 1W [6].

In summary the ARM processor's existing market share of the mobile phone market combined with an emphasis on small code size and low-power operation is the main reason for its widespread usage on Android phones. This is a strong contrast to most desktop/laptop/server Linux systems, which commonly use x86 processors instead.

3 KERNAL MODIFICATIONS

Android is based on the Linux, but does not use a standard Linux kernel. The kernel enhancements of Android include *alarm driver*, *ashmem* (*Android shared memory driver*), *binder driver*(Inter-Process Communication Interface), *power management*, *low memory killer*, *kernel debugger* and *logger*. All these kernel enhancements have been contributed back to the open source community under the GNU Public License (GPL) [8]. Here we list a summary of the most substantial components :

- *alarm driver*: provides timers to wake devices up from sleep
- *ashmem*: allows applications to share memory and manages the sharing in kernel levels.
- *binder driver*: facilitates inter-process communication since data can be shared by multiple applications through the use of shared memory. A service registered as an IPC service do not have to worry about different threads because binder will handle, monitor and manage them. Binder also takes care of synchronization between processes.
- *power management*: built on the top on standard Linux Power Management (PM) and take a more aggressive policy to manage and save power. This addition is described in detail in section 7.

4 BIONIC: STANDARD C LIBRARY

On most Linux distributions the GNU C library is used to provide the library routines specified by the ISO C standard [9] for C language programs. Many developers view the GNU C library as being inappropriate for memory constrained platforms such as embedded systems [10]. Furthermore, this library is licensed under the GNU Lesser Public License (LGPL) [11] and therefore restricts licensing of derivative works. These concerns led the developers of Android to instead create their own C library called "Bionic". This library was designed to have fast execution paths, avoid edge cases and remain a simple implementation. It is composed partly from the BSD C library combined with Android original source code. This results in a combination of the BSD and Android licenses covering the entire library.

Among the most complex aspects of the C library is its support for user level threads. In the GNU C library user level threads are provided by the Native POSIX Thread Library (NPTL) [13]. NPTL offers high performance POSIX threads which are especially beneficial in server applications. However, for a far more resource limited system, NPTL's memory and disk space footprints are far too large to be considered practical. Many of the supported features were not replicated due to the small number of concurrent processes running on Android. These include process shared mutexes and conditional variables. Inter-process communication support (IPC) was also deemed unnecessary due to the preexisting Android "Intents" service. Aditional features were removed due to their limited benefit for significant effort. Cancelation of threads in particular was cited as requiring extensive code to be properly supported. Likewise only one real-time timer is permitted per thread as opposed to standard Linux where the library attempts to optimize the threads in use for performance reasons. Similarly C++ exception support was not included to further reduce code space and minimize complexity [12].

Similar to sysctl [14] offered in BSD systems, Android provides access to kernel properties at run-time. This allows userspace programs to alter kernel behavior dynamically. Only processes with the appropriate permissions are allowed to modify these settings. The security of this system is maintained by assigning a unique user id and group id pair to each application. Since the target system is intended to be used by a single user, the typical /etc/passwd and /etc/groups settings have been removed. As a further security measure, /etc/services has also been replaced with a constant list of services maintained inside the executable itself.

The divergent goals of the Android development team drove them to create a custom implementation of the Standard C library. This library is especially suited to operate with the limited CPU and memory available on Android platforms. Furthermore, special security provisions were made in order to ensure the integrity of the system.

5 DALVIK VIRTUAL MACHINE

Many of the top cell phone manufacturers such as Nokia, Motorola and Samsung include a mobile optimized version of the Java virtual machine called Java 2, Micro Edition (J2ME). In contrast to these vendors Android uses their own Dalvik Virtual Machine (See Figure 1). This development process is identical to the developer as a standard Java platform. Application developers write their program in Java and compile java class files. However, instead of the class files being run on a J2ME virtual machine, the code is translated after compilation into a "Dex file" that can be run on the Dalvik machine. A tool called dx will convert and repack the class files in a Java .jar file into a single dex file with several shared constant pools. This is used to reduce the duplicated arguments and make the dex file more compact. The virtual machine itself is optimized to perform well on mobile devices with a slow CPU, limited memory, no operating system swap space and most importantly limited battery power [15]. These constraints can be quite tight. For example, typically the total system memory will be approximately 64M. High-level services will consume roughly 44MB. Then, the large system library will consume another 10MB. This leaves only 10MB for pure application code. The Dalvik VM is optimized for low memory compared to other standard VMs due to the following changes [16]:

- The VM was slimmed down to use less space.
- Dalvik has no just-in-time compiler
- The constant pool has been modified to use only 32-bit indexes to simplify the interpreter.
- It uses its own bytecode, not Java bytecode.

6 FILE SYSTEM

6.1 Storage media: NAND

Android uses the YAFFS flash file system, the first NAND optimized Linux flash file system. For mobile devices, hard disks are too large in size, too fragile and consume too much power to be useful. In contrast, flash memory provides fast read access time and better kinetic shock resistance than hard disks. There are fundamentally two different types of flash memory based on their construction technique: NOR and NAND. NOR is low density, offers slow writes and fast reads. NAND is low cost, high density and offers fast writes and slow reads. Embedded systems are increasingly using NAND flash for storage and NOR for code and execution [21]. File systems for flash memory have to deal with these limitations in order to provide a robust file system.

Limitations

Important limitations of NAND memory include block erasure and memory wear. Block erasure means that when erasing any memory the whole block must be erased. NAND can be randomly accessed on a page basis during programming, but cannot offer arbitrary random-access rewrite or erase during normal operation. To overcome this limitation, memory segments are marked to be removed or “dirty”. When the entire block is dirty, then it can be erased. Memory wear means that there is a limited number of erase-write cycles in the flash memory and at the end of its lifetime the data integrity of storage will deteriorate. Wear leveling

techniques are used to uniformly use whole sections and to optimize the total lifetime of the device. Bad block management (BBM) is also used to perform write verification and remapping to spare sectors in case of write failure.

6.2 File system for NAND: YAFFS

YAFFS (Yet Another Flash File System) was developed by Toby Churchill Ltd (TCL) as a reliable filing system with fast boot time for their flash memory devices. They initially tried to modify existing flash file systems such as JFFS (used mainly for NOR) to add NAND support, but it turned out that the slow boot time and RAM consumption of existing flash file systems was unacceptable. Furthermore, there are too many fundamental differences between NOR and NAND to make performance optimal. For instance, since erasing NOR is much longer than for NAND, garbage collection methodologies for NOR are not suitable for NAND. This led them to develop a different flash file system especially for NAND according to its features and limitations to optimize performance and ensure robustness. Upon completion YAFFS performed better than existing flash file systems and can still be used with NOR flash even though it is was specifically designed for NAND.

YAFFS is the first flash file system specifically designed for NAND flash. It is highly portable and has been used on Linux, WinCE, pSOS, eCos, ThreadX and various special-purpose operating systems. Compared to general purpose disk file systems, flash file systems eliminate seeking times, but they have to deal with lifetime limitations and error correction. Mobile devices using YAFFS can usually tolerate far less errors in file system because the system can easily crash due to a corrupted file system. To avoid this problem, YAFFS provides bad block management and error correction to maintain data integrity in NAND flash and achieve a fast robust file system. The schema of YAFFS can be seen in Figure 2 .

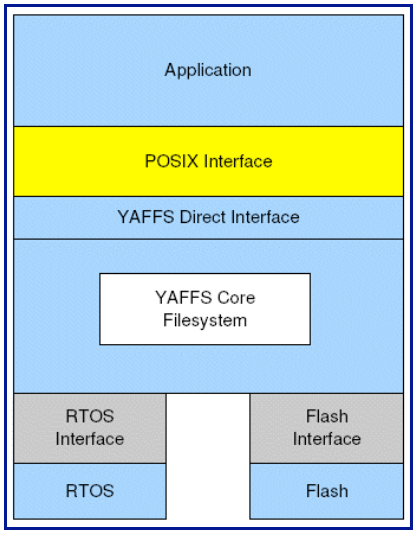

Figure 2 YAFFS embedded structure: YAFFS embedded structure [20]

YAFFS includes the following features:

- *Journaling*: a log-structured file system, which makes it robust to possible power failures. Note that YAFFS requires some RAM to maintain the data structure for its log.
- *Garbage collection*: highly optimized and predictable garbage collection strategies that

makes it high performance and deterministic under hard writing. Collection is executed when free space becomes low. A block with some dirty pages and some good pages will be chosen, and YAFFS will move those good pages to another block. Then the block will be marked as dirty and erased.

- *Lower memory requirement*: it has a lower memory footprint than most other log-structured flash file systems, so it is more scalable.
- *Flexibility*: it uses a more general definition of NAND flash, is highly configurable and can be customized to work with various flash devices, different geometries, different error correction options, caching options, etc.
- *Portability*: although YAFFS was originally designed for use with Linux, its modular design make it easy and portable to many operating systems. The file-system-specific code was kept separate from the main YAFFS file system code. The YAFFS Direct Interface (YDI) can hook up to other filesystems by applying a wrapper layer.
- *Robustness*: it is well tested and has been used in many products. Bad block handling and error correction mitigate the limitations of flash memory.
- *POSIX Support*: it provides a wide range of POSIX-style file system support including directories, symbolic and hard links through standard file system interface calls.

We can conceptually treat YAFFS as an improved version of flash file system since it outperforms previous flash file system in many ways. The main differences between YAFFS and its predecessor JFFS are [18]:

1. YAFFS uses less run-time RAM.
2. YAFFS garbage collection is simpler and faster.
3. YAFFS uses a whole page per file for headers and provides no compression.
4. YAFFS can be used on NOR but, will not perform very well. Therefore, partition size can be a guideline to choose between YAFFS and JFFS. For smaller partition sizes JFFS is better suited whereas for larger sizes YAFFS performs better.

6.3 Comparison with other file system

A standard Linux system typically does not use flash memory, but rather magnetic disk drives. These drives are commonly formatted by default to use the latest version of the Ext file system. As of this writing, Ext3 is currently in widespread use and is presented here for comparison with the flash filesystem, YAFFS, in use on Android.

Ext3

Third extended file system, commonly known as Ext3, is also a journaled file system used by the Linux kernel. It is more reliable than its predecessor, Ext2, through the use of journaling to facilitate fast reboots after a system crash. This means that Ext3 does not have to operate a whole file system check after an unclean shutdown, as was previously done using *fsck*. The main goal of Ext3 was to create a journaling file system which is backward compatible with Ext2. Ext3 enhances its reliability by using a special API called the Journaling Block Device layer (JDB) to physically journal both metadata and data (when the level of journaling is configured as the "journal mode"). It also uses complete physical blocks to backup and store modified blocks instead of only recording spans of bytes. Ext3 has three levels of journaling as mounting options, which vary the trade-off between data integrity and performance:

1. *Journal mode*: performs full data and meta data journaling before committing to the file system to achieve the highest data integrity and lowest risk among the three modes. However, full journaling can be slow in performance.
2. *Ordered mode (Default)*: journals metadata only. "Ordered" means that when the file system is about to write to the disk, the data blocks will be written to the disk first, and then the metadata block for the data will be committed to the file system.
3. *Writeback mode*: only metadata blocks are written to the journal, but does not guarantee the write order of data blocks and metadata. It relies on the standard file system write process to write file data changes back to disk. This mode has the highest risk but, operates fastest among the three options.

YAFFS-Ext3 Comparison
YAFFS (flash file system) and Ext3 (disk file system) have fundamental differences with each other, since the design principles of these two file system are based on the physical differences of each storage media. These differences can be summarized as follows:

- *File accessibility*: disk file systems are optimized to avoid disk seeks whenever possible due to high seeking cost; flash memory devices have no seek latency and can randomly access files.
- *Block erasing*: it is easy to erase a file on a disk; for flash devices it is quite time consuming therefore it should be done while the device is idle.
- *Wear leveling techniques*: only flash file systems have to deal with limited lifetime.

7 POWER MANAGEMENT

Power management in operating systems is necessary due to the ever increasing power demand of modern desktop computers and especially laptops. In order to reduce wasted power, multiple hardware power saving features are employed by Linux such as clock gating, voltage scaling, activating sleep modes and disabling memory cache. Each of these features reduces the system's power consumption at the expense of latency and/or performance. These tradeoffs on a Linux system are managed by either Advanced Power Management (APM) or Advanced Configuration and Power Interface (ACPI). APM is an older, simpler, BIOS based power management subsystem, which is still used on older systems. Newer systems use ACPI based management instead. ACPI is more operating-system centric [25] than APM and also offers more features such as a tree structure for powering down devices so that subsystem components are not turned off before the subsystem itself.

In contrast with a standard Linux system, Android does not use APM, nor ACPI for power management. As shown in Figure 1, Android instead has its own Linux power extension, PowerManager instead. The core power driver (Shown at the bottom of Figure 3 as "Power") was added to the Linux kernel in order to facilitate this functionality. This module provides low level drivers in order to control the peripherals supported by the Power Manager. These peripherals currently include: screen display and backlight, keyboard backlight and button backlight. Each peripheral's power is controlled through the use of WakeLocks. These locks are requested through the API whenever an application requires one of the managed peripherals to remain powered on (Each lock setting shown in Table 1). If no wake lock exists which "locks" the device, then it is powered off to conserve battery life. In the case of multiple power settings the transition is managed through the use of delays based on system activity. A sample of this behavior is shown in Figure 4 for the screen backlight. In addition to WakeLocks the Power

Manager also monitors the battery life and status of the device. This service coordinates with the power circuitry charging in the battery and also powers down the system when the battery reaches a critical threshold.

Table 1 Wake Lock Settings [26]

Flag Value	CPU	Screen	Keyboard
PARTIAL_WAKE_LOCK	On*	Off	Off
SCREEN_DIM_WAKE_LOCK	On	Dim	Off
SCREEN_BRIGHT_WAKE_LOCK	On	Bright	Off
FULL_WAKE_LOCK	On	Bright	Bright

**If you hold a partial wakelock, the CPU will continue to run, irrespective of any timers and even after the user presses the power button. In all other wakelocks, the CPU will run, but the user can still put the device to sleep using the power button.*

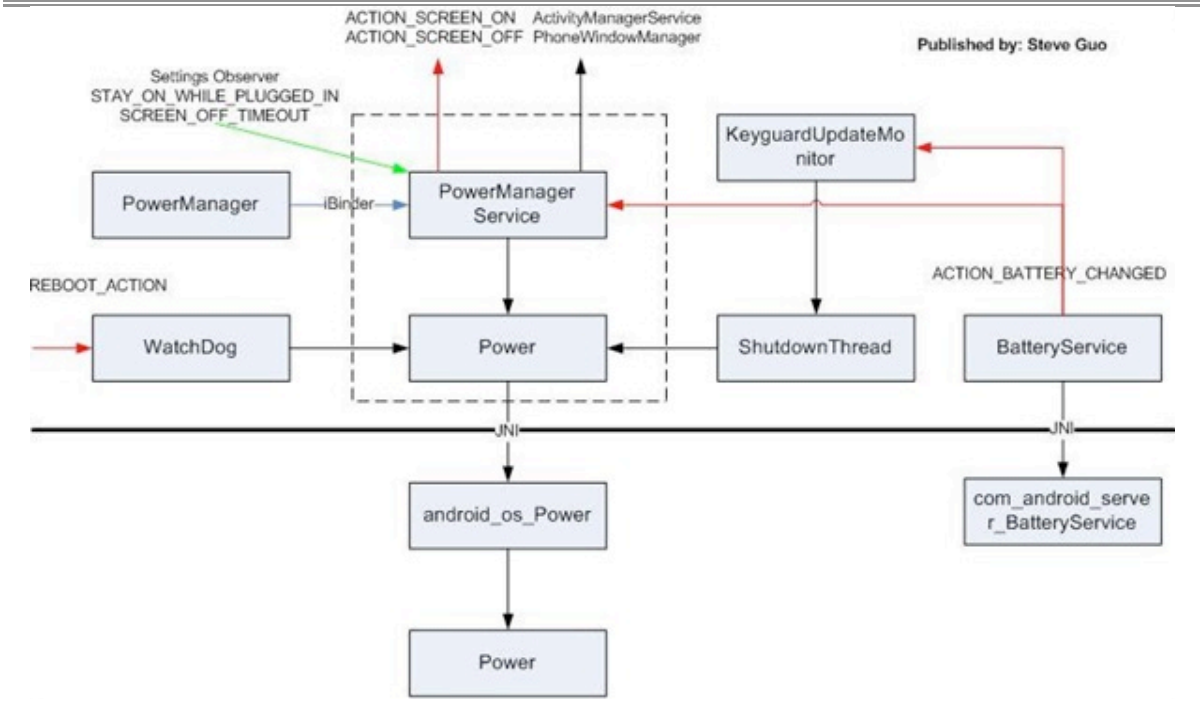

Figure 3 Android Power Management [24]

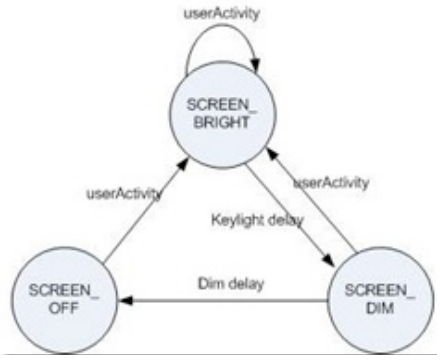

Figure 4 Power Management State Machine (Screen Brightness) [24]

8 CONCLUSION

In conclusion, we compared the major differences between Android for mobile devices and Linux for desktop/laptops/servers systems. We have examined Android in a bottom-up fashion from its architecture to obtain an extensive overall understanding. We focused on the following general topics to clarify the design details of Android: architecture, kernel design, standard C Library (including user threads), Java virtual machine, file system and power management. In order to understand how Android overcomes the hardware limitations of a mobile device as a light-weight operating system, we depicted and summarized the key features of Android with a reasonable level-of-details in each topic.

References

[1] [TA6] Androidology I: Architecture http://www.android.com/about/videos.html#video=androidologyiarchitecture

[2] [TA1] Gartner Says Worldwide Smartphone Sales Reached Its Lowest Growth Rate With 3.7 Per Cent Increase in Fourth Quarter of 2008, http://www.gartner.com/it/page.jsp?id=910112

[3] [TA2] ARMed for the living room, http://news.cnet.com/ARMed-for-the-living-room/2100-1006_3-6056729.html

[4] [TA3] Improving ARM Code Density and Performance, http://www.arm.com/pdfs/Thumb-2CoreTechnologyWhitepaper-Final4.pdf

[5] [TA4] ARM System-on-Chip Architecture, Steve Furber, Addison-Wesley Professional

[6] [TA5] Intel Atom Processor, http://www.intel.com/technology/atom/

[7] [KM1] Anatomy & Physiology of an Android, Google I/O Developer Conference 2008, http://sites.google.com/site/io/anatomy--physiology-of-an-android

[8] [KM2] Source repository of Android Kernel, http://git.android.com

[9] [SCL2] http://en.wikipedia.org/wiki/C_standard_library

[10][SCL1] http://en.wikipedia.org/wiki/GNU_C_Library

[11][SCL3] http://en.wikipedia.org/wiki/GNU_Lesser_General_Public_License

[12][SCL4] http://osdir.com/ml/android-platform/2009-02/txtRWirPcEReT.txt

[13][SCL5] http://people.redhat.com/drepper/nptl-design.pdf

[14][SCL6] http://www.openbsd.org/cgi-bin/man.cgi?query=sysctl

[15][VM0] Dalvik VM Internals, Google I/O Developer Conference 2008,

[16][VM1] Dalvik virtual machine on wikipedia http://en.wikipedia.org/wiki/Dalvik_virtual_machine

[17][VM2] Dalvik overview http://www.dalvikvm.com/

[18][FS1] YAFFS: the NAND-specific flash file system - Introductory Article http://www.yaffs.net/yaffs-nand-specific-flash-file-system-introductory-article

[19][FS2] YAFFS at Wikipedia http://en.wikipedia.org/wiki/YAFFS

[20][FS3] YAFFS structure http://www.yaffs.net/yaffs-overview

[21][FS4] NAND vs. NOR Flash Memory: Technology Overview, http://www.toshiba.com/taec/components/Generic/Memory_Resources/NANDvsNOR.pdf

[22]http://sites.google.com/site/io/dalvik-vm-internals

[23] [PM1] Everything You Need to Know About the CPU C-States Power Saving Modes

http://www.hardwaresecrets.com/article/611/1
[24][PM2] Android Power Management
http://letsgoustc.spaces.live.com/Blog/cns!89AD27DFB5E249BA!526.entry
[25][PM3] Advanced Configuration and Power Interface
http://en.wikipedia.org/wiki/Advanced_Configuration_and_Power_Interface
[26][PM4] Android SDK: PowerManager
http://developer.android.com/reference/android/os/PowerManager.html