Notes

* Process Scheduling
  + IOS
    - Same as MAC
    - OS X and IOS Kernel Programming
  + OS 360/370
    - So, FIFO scheduling or non-preemptive priority scheduling.
    - The job scheduler is responsible for job management functions. It permits either sequential FIFO scheduling or non-preemptive priority scheduling. Priority scheduling is accomplished via an input work queue.
    - IBM OS/360: An Overview of the First General Purpose Mainframe under Job and Task Management
    - Therefore, the OS implemented an arbitrary absolute limit of 30 minutes before canceling the job, this freeing the system for the next job to begin processing.
  + Windows
    - Same as MAC and UNIX
    - Threads are scheduled to run based on their scheduling priority. Each thread is assigned a scheduling priority. The priority levels range from zero (lowest priority) to 31 (highest priority).
    - The system treats all threads with the same priority as equal. The system assigns time slices in a round-robin fashion to all threads with the highest priority. If none of these threads are ready to run, the system assigns time slices in a round-robin fashion to all threads with the next highest priority. If a higher-priority thread becomes available to run, the system ceases to execute the lower-priority thread (without allowing it to finish using its time slice), and assigns a full time slice to the higher-priority thread.
    - https://msdn.microsoft.com/en-us/library/windows/desktop/ms685100(v=vs.85).aspx
  + Mac OS X
    - Round Robin mixed with highest priority
    - In its typical operation, the Mac OS X scheduler gives the processor to each thread for a brief period of time, after which it considers switching to another thread. The amount of time a scheduled thread can run before being preempted is called the thread's timslicing quantum, or simply quantum. Once a thread's quantum expires, it can be preempted because another thread of equal or higher priority wants to run. Moreover, a running thread can be preempted regardless of its quantum if a higher-priority thread becomes runnable.
      * Mac OS X Internals section 7.4
  + Android
    - Completely Fair Scheduler
    - Android operating system uses O (1) scheduling algorithm as it is based on Linux Kernel 2.6. Therefore the scheduler is names as Completely Fair Scheduler as the processes can schedule within a constant amount of time, regardless of how many processes are running on the operating system
      * “Priority based preemptinve task article”
  + Unix
    - The scheduler on the UNIX system belongs to the general class of operating system schedulers known as round robin with multilevel feedback, meaning that the kernel allocates the CPU to a process for a time quantum, preempts a process that exceeds its time quantum, and feeds it back into one of several priority queues.
    - At the conclusion of a context switch, the kernel executes the algorithm to schedule a process (Figure 8.1), selecting the highest priority process from those in the states "ready to run and loaded in memory" and "preempted." It makes no sense to select a process if it is not loaded in memory, since it cannot execute until it is swapped in. If several processes tie for highest priority, the kernel picks the one that has been "ready to run" for the longest time, following a round robin scheduling policy
      * Design of unix operating system page 248
  + Multics
    - The choice of which processes to postpone depends on several factors. If some processes have higher priority than others, the lower priority processes will be postponed. If, in the lowest priority class which will continue to run, some processes have been prescheduled for given completion times or computing rates, the prescheduled processes will be given preference. Finally, to make a choice among processes otherwise equal, the scheduler will prefer a process currently using expensive facilities (e.g., core) over one occupying inexpensive facilities (e.g., drum); the former is in some sense using more system resources than the latter, so it is desirable to move it toward completion.
    - The first Multics scheduler, circa 1967, was descended from the one in use on CTSS. It was sometimes described as a "Greenberger-Corbató exponential scheduler." The scheduler had N queues, something like 6. When the scheduler wanted to choose a process to run, it took the one at the head of the highest priority empty queue. A process that started requesting service would be put at the tail of the highest priority queue. Processes from that queue were given a small time slice, called the QUANTUM. If the timer went off and the process was still running, it was sent to the tail of the next lower queue, and when processes from that queue were run, they got a time slice twice as long (thus, exponentially longer time slices). The lowest priority queue was called the "background" queue. Thus, long-running jobs would not run until higher priority jobs were done; the scheduler would then run these background jobs for long blocks of time. A job that blocked on disk I/O was skipped over but didn't lose its position in queue; a job that blocked on terminal I/O started over at the top queue when it received an input wakeup. There are more details, but that's a general idea. [THVV]
    - In 1975, Bob Mullen implemented the Multics "Workclass" scheduler, originally done to provide different percentages of the machine to different groups of users. It was not that each user got a few percent, but that a group was to get some total percent. (Scheduling of time for users within the group was done by what is called an FB-n algorithm, meaning n levels of foreground and background priorities.)
    - This scheduler was overlaid on the original FB-n scheduler, in the sense that scheduling within a group was done by the FB-n scheduler. The workclass scheduler decided to make a process "eligible" based on its workclass being the most worthy; the ready queue for each workclass was sorted as an FB-n queue.
* Memory Management
  + Unix
    - Historically, UNIX systems transferred entire processes between primary memory and the swap device, but did not transfer parts of a process independently, except for shared text. Such a memory management policy is called swapping. It made sense to implement such a policy on the PD? 11, where the maximum process size was 64K bytes. For this policy, the size of a process is bounded by the amount of physical memory available on a system. The BSD system (release 4.0) was the first major implementation of a demand paging policy, transferring memory pages instead of processes to and from a secondary device;
    - Page 271
  + IBM OS 360
    - In its original incarnation, PCP (Primary Control Program), OS/360 operated as a batch system, and thus, one task had access to all resources on the system. Thus, the memory model used at this time was a single partition. The OS existed in a portion of main memory and the remaining available memory was used by the task that was currently existing on the system.
    - Another option for the OS was MFT, multiprocessing with a fixed number of tasks. Recall that this option allowed from 4 – 15 tasks to exist at a time. This option utilized memory that was divided into Multiple Fixed Partitions. If a task was idle, that partition remained held by the task until it was completed.
    - In the most complex implementation, multiprogramming with a variable (limitless) number of tasks (MVT), partitions were created on the fly. If memory was free, the control module would search the job queue for a job that would fit 8 into the available space and would create a partition for that task. For this reason, external fragmentation was a problem. An enhancement addressed the fragmentation issue by tagging each task in the ready queue with one of several fixed size tags. When there jobs were released they were fit into predefined slots. This of course would result in internal fragmentation but had the advantage of allowing a very large job to find room on the system, where it might not be able to in the previous incarnation of the scheduler
    - Page 8-9
  + Android
    - The Android Runtime (ART) and Dalvik virtual machine use [paging](http://en.wikipedia.org/wiki/Paging) and [memory-mapping](http://en.wikipedia.org/wiki/Memory-mapped_files)(mmapping) to manage memory. This means that any memory an app modifies—whether by allocating new objects or touching mmapped pages—remains resident in RAM and cannot be paged out. The only way to release memory from an app is to release object references that the app holds, making the memory available to the garbage collector. That is with one exception: any files mmapped in without modification, such as code, can be paged out of RAM if the system wants to use that memory elsewhere.
    - Out-of-memory killer
      * To address this, Android introduces its own out-of-memory killer to the kernel, with different semantics and design goals. The Android out-of-memory killer runs much more aggressively: whenever RAM is getting ‘‘low.’’ Low RAM is identified by a tunable parameter indicating how much available free and cached RAM in the kernel is acceptable. When the system goes below that limit, the out-of-memory killer runs to release RAM from elsewhere. The goal is to ensure that the system never gets into bad paging states, which can negatively impact the user experience when foreground applications are competing for RAM, since their execution becomes much slower due to continual paging in and out.
  + Windows
    - Although much of the memory-management system is invisible to programmers, one important feature is visible: namely the ability of a process to map a file onto a region of its virtual memory. This allows threads running in a process the ability to read and write parts of the file using pointers without having to explicitly perform read and write operations to transfer data between the disk and memory. With memory-mapped files the memory-management system itself performs the I/Os as needed (demand paging). Windows implements memory-ma
      * Page 873 Modern Operating System
    - Windows, on the x86, supports a single linear 4-GB demand-paged address space per process. Segmentation is not supported in any form.
      * P. 933 MOS
  + Multics
    - In Multics (**Mult**iplexed **I**nformation and **C**omputing **S**ervice) segmentation provides a generalized basis for the direct accessing and sharing of on line information
* User Interface
  + Unix
    - It has a simple user interface that has the power to provide the services that users want.
    - Most of the common personal computer distributions of Linux have replaced this keyboard-oriented user interface with a mouse-oriented graphical user interface, without changing the operating system itself at all. It is precisely this flexibility that makes Linux so popular and has allowed it to survive numerous changes in the underlying technology so well.
    - Most UNIX systems use the X Window System as the basis of the user interface. It consists of programs that are bound to special libraries that issue drawing commands and an X server that writes on the display
    - Shell
      * Although Linux systems have a graphical user interface, most programmers and sophisticated users still prefer a command-line interface, called the shell. Often they start one or more shell windows from the graphical user interface and just work in them. The shell command-line interface is much faster to use, more powerful, easily extensible, and does not give the user RSI from having to use a mouse all the time. Below we will briefly describe the bash shell (bash). It is heavily based on the original UNIX shell, Bourne shell (written by Steve Bourne, then at Bell Labs). Its name is an acronym for Bourne Again SHell. Many other shells are also in use (ksh, csh, etc.), but bash is the default shell in most Linux systems.
      * P. 726 MOS
    - P. 729 MOS
  + Android
    - UI elements in Android include traditional widgets such as buttons, text boxes, dialogs, menus, and event handlers. This part of the API is relatively straightfor‐ ward, and developers usually find their way around it fairly easily if they’ve already coded for any other UI framework.
    - All UI objects in Android are built as descendants of the View class and are organized within a hierarchy of ViewGroups. An activity’s UI can actually be specified either statically in XML (which is the usual way) or declared dynamically in Java. The UI can also be modified at runtime in Java if need be. An activity’s UI is displayed when its content is set as the root of a ViewGroup hierarchy.
  + Windows
    - (in contrast to Windows, where even the GUI (Graphical User Interface) is in the kernel)
      * Pg 719 MOS
  + Multics
    - Ease of accessibility, featuring a simple and consistent user interface for all types of services. There is no job control or command language to learn and an interactive tutorial facility is available when needed
  + OS 360
    - There were two model-independent interfaces specified for the 360 line. The first interface was the instruction set; the second was the channelto-control-unit electrical specification and signaling protocols. The latter was standardized so that a single set of peripheral devices could be designed to work across the entire range of models. T.he channel-to-control-unit interface has some interesting and important features: It allows I/O equipment to be interchanged between systems; it also has some effect on the software, since the commands and the status indications are defined to be consistent. This makes a compatible or common programming system for I/O control possible. The channelto-control-unit interface was made public at the same time as the 360 Principles of Operation..
      * IBM Case Study P. 294
* Security
  + IBM OS 360
    - User management in this mainframe OS was virtually non-existent and thus the task of safeguarding sensitive data is limited to the use of a password. A data set can be flagged as “protected”. The correct password must then be entered on the console. Passwords are stored is a control table that has its own security flag set to “protected” can only be reached via the master password.
    - Page 11
  + Android
    - Your device must conform to the security environment enforced by the Android ap‐ plication framework, Dalvik, and the Linux kernel. Specifically, apps must have access and be submitted to the permission model described as part of the SDK’s documentation. Apps must also be constrained by the same sandboxing limitations they have by running as separate processes with distinct user IDs (UIDs) in Linux. The filesystem access rights must also conform to those described in the developer documentation. Finally, if you aren’t using Dalvik, whatever VM you use should impose the same security behavior as Dalvik.
    - Security in Android is enforced at the process level. In other words, Android relies on Linux’s existing process isolation mechanisms to implement its own policies. To that end, every app installed gets its own UID and group identifier (GID). Essen‐ tially, it’s as if every app is a separate “user” in the system. And as in any multiuser Unix system, these “users” cannot access one another’s resources unless permissions are explicitly granted to do so. In effect, each app lives in its own separate sandbox.
    - To exit the sandbox and access key system functionality or resources, apps must use Android’s permission mechanisms, which require developers to statically declare the permissions needed by an app in its manifest file. Some permissions, such as the right to access the Internet (i.e., use sockets), dial the phone, or use the camera, are predefined by Android. Other permissions can be declared by app developers 30 | Chapter 2: Internals Primer and then be required for other apps to interact with a given app’s components. When an app is installed, the user is prompted to approve the permissions required to run an app.
    - Access enforcement is based on per-process operations and requests to access a specific URI (universal resource identifier), and the decision to grant access to a specific functionality or resource is based on certificates and user prompts. The certificates are the ones used by app developers to sign the apps they make available through Google Play. Hence, developers can restrict access to their apps’ functionality to other apps they themselves created in the past.
    - All page 30
  + Mac OS X and iOS
    - For security, because a kernel extension is granted the same elevated privileges as the core operating system code, kernel extensions can only be installed or loaded by a user with administrative access to the system. As a further security measure, the system has strict requirements regarding the file permissions of the kernel extension’s bundle and will refuse to load a kernel extension that does not meet these requirements, (page 46)
    - Uses Unix security model (page 33)
  + Unix
    - The user community for a Linux system consists of some number of registered users, each of whom has a unique UID (User ID). A UID is an integer between 0 and 65,535. Files (but also processes and other resources) are marked with the SEC. 10.7 SECURITY IN LINUX 799 UID of their owner. By default, the owner of a file is the person who created the file, although there is a way to change ownership. Users can be organized into groups, which are also numbered with 16-bit integers called GIDs (Group IDs). Assigning users to groups is done manually (by the system administrator) and consists of making entries in a system database telling which user is in which group. A user could be in one or more groups at the same time. For simplicity, we will not discuss this feature further. The basic security mechanism in Linux is simple. Each process carries the UID and GID of its owner. When a file is created, it gets the UID and GID of the creating process. The file also gets a set of permissions determined by the creating process. These permissions specify what access the owner, the other members of the owner’s group, and the rest of the users have to the file. For each of these three categories, potential accesses are read, write, and execute, designated by the letters r, w, and x, respectively. The ability to execute a file makes sense only if that file is an executable binary program, of course. An attempt to execute a file that has execute permission but which is not executable (i.e., does not start with a valid header) will fail with an error. Since there are three categories of users and 3 bits per category, 9 bits are sufficient to represent the access rights.
    - Page 798-799
    - Password Security Generally speaking, the UNIX passwd program places very few restrictions on what may be used as a password. It is important to follow the MnSCU standard when choosing UNIX password parameters. Some implementations may or may not have the ability follow the MnSCU password standard. In these cases, a third-party software may be available or the system administrators may need to enforce passwords strength through security awareness training and security reminders.
    - Account Security Administration
      * Regularly audit your system for dormant accounts and disable any that have not been used for a specified period of time, in accordance with MnSCU's security standard.
      * Verify that all accounts have passwords. Check shadow, NIS, and NIS+ passwords to verify the password field is not empty.
    - Special Accounts
      * Ensure that there are no shared accounts (other than root) on high security systems (in accordance MnSCU standards. July 5, 2006 Page 2 of 13
      * Disable guest accounts and/or do not create guest accounts. Some systems come preconfigured with guest accounts.
      * Use special groups to restrict which users can use su to become root.
      * Disable all default vendor accounts shipped with the Operating System. This should be checked after each upgrade or installation.
      * Disable or delete accounts that have no password which execute a command, for example "sync". Delete or change ownership of any files owned by these accounts. Ensure that these accounts do not have any cron or at jobs.
      * Do not assign non-functional shells (such as /bin/false) to system accounts such as bin and daemon and to the sync account if it is not needed.
    - Root Account
      * Restrict the number of people who know the root password. These should be the same users registered with groupid 0.
      * Do not log in as root over the network, use su instead. This provides greater tracking and accountability.
      * Verify that root does not have a ~/.rhosts file.
      * Verify that "." is not in root's search path.
  + Multics
    - On Multics, as on many systems, the first line of defense is a set of tables which lists users and their access rights to data. These tables are scanned by the operating system on each user's reference to a block of data. In theory this is a simple and unbreachable defense. In practice it is often very vulnerable, for three reasons:
    - The hardware architecture may contain exploitable behavior (or misbehavior). For example, the hardware implementation may offer opportunities for trap doors, which can be opened under specific conditions.
    - The software utilization of the table look-up mechanisms may contain exploitable errors.
    - The table mechanism may be completely circumvented by implementation errors in the system's operating software.
    - On Multics, the table used to determine access is the **Access Control List, or ACL**, associated with each block of data, or segment (file), in the system.
    - However, the extended access control system used on some Multics systems, **AIM, or Access Isolation Mechanism**, to a large extent solves the security problem posed by users with access to high-level directories by increasing the number of attributes of each segment and each user, and by enforcing a stricter set of rules for matches between the two.
  + Windows
    - The security reference monitor enforces Windows’ elaborate security mechanisms, which support the international standards for computer security called Common Criteria, an evolution of United States Department of Defense Orange Book security requirements. These standards specify a large number of rules that a conforming system must meet, such as authenticated login, auditing, zeroing of allocated memory, and many more. One rules requires that all access checks be implemented by a single module within the system. In Windows, this module is the security reference monitor in the kernel. We will study the security system in more detail in Sec. 11.10.
    - The new security model does not grant administrative privileges at all times. Even administrators run under standard privileges when they perform non-administrative tasks that do not require elevated privileges. The result is greater security because users are no longer running with unnecessary privileges that can be maliciously exploited. This feature is known as User Access Control, or UAC.
  + Ubuntu
    - By default, Ubuntu requires a minimum password length of 6 characters,
      * Can set password age
    - Ubuntu developers made a conscientious decision to disable the administrative root account by default in all Ubuntu installations. This does not mean that the root account has been deleted or that it may not be accessed. It merely has been given a password which matches no possible encrypted value, therefore may not log in directly by itself.
    - Instead, users are encouraged to make use of a tool by the name of *sudo* to carry out system administrative duties. *Sudo* allows an authorized user to temporarily elevate their privileges using their own password instead of having to know the password belonging to the root account. This simple yet effective methodology provides accountability for all user actions, and gives the administrator granular control over which actions a user can perform with said privileges.
    - Anyone that has physical access to the keyboard can simply use the **Ctrl**+**Alt**+**Delete** key combination to reboot the server without having to log on. While someone could simply unplug the power source, you should still prevent the use of this key combination on a production server. This forces an attacker to take more drastic measures to reboot the server, and will prevent accidental reboots at the same time.
    - The Linux kernel includes the *Netfilter* subsystem, which is used to manipulate or decide the fate of network traffic headed into or through your server. All modern Linux firewall solutions use this system for packet filtering.
    - The kernel's packet filtering system would be of little use to administrators without a userspace interface to manage it. This is the purpose of iptables: When a packet reaches your server, it will be handed off to the Netfilter subsystem for acceptance, manipulation, or rejection based on the rules supplied to it from userspace via iptables. Thus, iptables is all you need to manage your firewall, if you're familiar with it, but many frontends are available to simplify the task.
* File System
  + Unix
    - Bottom of Page 6, downwards
    - As observed in Chapter 2, every file on a UNIX system has a unique mode. The Mode contains the information necessary for a process to access a file, such as file ownership, access rights, file size, and location of the file's data in the file system. Processes access files by a well defined set of system calls and specify a file by a character string that is the path name. Each path name uniquely specifies a file, and the kernel converts the path name to the file's mode
    - Each path name uniquely specifies a file, and the kernel converts the path name to the file's mode
    - The kernel identifies particular modes by their file system and mode number and allocates in-core modes at the request of higher-level algorithms.
      * P. 60 Unix Design
  + MAC OS X and iOS
    - The kernel has inbuilt support for a range of different file systems, as shown in Table 2-7. The primary file system used by Mac OS X and iOS is HFS+. P. 34
    - HFS+ The standard file system used by Mac OS X and iOS
    - HFS Legacy Mac OS file system
    - UFS The BSD Unix file system
    - NFS Networked File System
    - ISO 9660 and UDF Standard file systems used by CDs and DVDs
    - SMB Server Message Block, a networked file system used to connect with Microsoft Windows computers
    - AFP Apple Filing Protocol
    - HFS+ gained support for journaling in Mac OS X 10.2.2. Journaling improves the reliability of a file system by recording transactions in a journal prior to carrying them out. This makes the file system resilient to events such as a power failure or a crash of the kernel, as the data can be replayed after reboot in order to bring the file system to a consistent state.
    - HFS+ supports very large files, up to 8 EiB in size (1 Exbibyte = 260 bytes), which is also the maximum possible volume size. The file system has full support for Unicode characters in file names and is case insensitive by default. Support for both Unix style file permissions and access control lists (ACLs) exists.
    - OS X provides “out-of-the-box” support for several different file systems. These include Mac OS Extended format (HFS+), the BSD standard file system format (UFS), NFS(an industry standard for networked file systems), ISO 9660 (used for CD-ROM), MS-DOS, SMB (Windows file sharing standard), AFP (Mac OS file sharing), and UDF.
    - Support is also included for reading the older, Mac OS Standard format (HFS) file-system type; however, you should not plan to format new volumes using Mac OS Standard format. OS X cannot boot from these file systems, nor does the Mac OS Standard format provide some of the information required by OS X.
    - UFS provides case sensitivity and other characteristics that may be expected by BSD commands. In contrast, Mac OS Extended Format is not case-sensitive (but is case-preserving).
    - OS X currently can boot and “root” from an HFS+, UFS, ISO, NFS, or UDF volume. That is, OS X can boot from and mount a volume of any of these types and use it as the primary, or root, file system.
    - NFS provides access to network servers as if they were locally mounted file systems. The Carbon application environment mimics many expected behaviors of Mac OS Extended format on top of both UFS and NFS. These include such characteristics as Finder Info, file ID access, and aliases.
    - By using the OS X Virtual File System (VFS) capability and writing kernel extensions, you can add support for other file systems. Examples of file systems that are not currently supported in OS X but that you may wish to add to the system include the Andrew file system (AFS) and the Reiser file system (ReiserFS). If you want to support a new volume format or networking protocol, you’ll need to write a file-system kernel extension.
      * Kernel Programming Guide
  + Windows
    - Windows supports several file systems, the most important of which are FAT-16, FAT-32, and NTFS (NT File System). FAT -16 is the old MS-DOS file system. It uses 16-bit disk addresses, which limits it to disk partitions no larger than 2 GB. Mostly it is used to access floppy disks, for those customers that still use them. FAT-32 uses 32-bit disk addresses and supports disk partitions up to 2 TB. There is no security in FAT -32 and today it is really used only for transportable media, like flash drives. NTFS is the file system developed specifically for the NT version of Windows. Starting with Windows XP it became the default file system installed by most computer manufacturers, greatly improving the security and functionality of Windows. NTFS uses 64-bit disk addresses and can (theoretically) support disk partitions up to 2^64 bytes, although other considerations limit it to smaller sizes.
      * P. 954 MOS
  + OS 360/370
    - The word "file" is generally not used in the Operating System/360 publications, to avoid possible ambiguity with the auxiliary storage devices on which the data sets appear.
    - To retrieve a data set, the system needs the data set name, the volume serial number, the device type, and in some instances, the data set sequence number. Specifying these can sometimes be inconvenient for the programmer. The catalog permits storage and retrieval of a data set based on name alone.
    - A data set name consists of a simple name preceded by qualifiers (index names) separated by periods. Each component of the full name can be up to 8 characters long; the entire name, including periods, can be up to 44 characters long. Each component name starts with an alphabetic character, and may contain any letter or number. Indexes, linked together in a hierarchy specified by the user, contain pointers to subordinate indexes, to volumes, or to both.
  + Android
    - **exFAT** - The extended File Allocation Table is a Microsoft proprietary file system for flash memory. Due to the licensing requirements, it is not part of the standard Linux kernel. However, some manufactures provide Android support for the file system.
    - **F2FS** - Samsung introduced The Flash-Friendly File System as an open source Linux file system in 2012.
    - **JFFS2** - The Journal Flash File System version 2 is the default flash file system for the AOSP (Android Open Source Project) kernels, since Ice Cream Sandwich. JFFS2 is a replacement to the original JFFS.
    - **YAFFS2** - Yet Another Flash File System version 2 was the default AOSP flash file system for kernel version 2.6.32. YAFFS2 is not supported in the newer kernel versions, and does not appear in the source tree for the latest kernel versions from kernel.org. However, individual mobile device vendors may continue to support YAFFS2

Besides flash memory file systems, Android devices typically support the following media-based file systems:

* **EXT2** / **EXT3** / **EXT4** - The EXTended file system is the standard Linux file system, with EXT4 being the current version. Since 2010, EXT4 is often used in place of YAFFS2 or JFFS2 as the file system for internal flash memory on Android devices.
* **MSDOS** - The MSDOS driver supports the FAT12, FAT16 and FAT32 file systems.
* **FAT** - The VFAT is not actually a file system, but an extension to the FAT12, FAT16, and FAT32 file systems. Thus, you will always see the VFAT kernel module in conjunction with the MSDOS module. External SD Cards are commonly formatted using VFAT.

The above file systems are media-based file systems. VFS also supports pseudo file systems, which are not media based. The Linux kernel supports a number of pseudo file systems, the ones that are important to Android devices are:

* **cgroup** - The cgroup (control group) pseudo file system provides a means to access, and define, various kernel parameters. While cgroup is the pseudo file system, there are a number of different process control groups. If your Adroid device supports process control groups, you will find a list of the groups in the file */proc/cgroups*. Android uses cgroups for acct (user accounting), and cpuctl (CPU control).
* **rootfs** - This file system serves as the mount point for the root file system ("/").
* **procfs** - The procfs file system is normally mounted on the */proc* directory, and reflects a number of kernel data structures. The  operations on these files actually read live kernel data. The number directories reflect the process IDs (actually, the thread group leader process ID) for each running task. The */proc/filesystems* file generates a list of the currently registered file systems. File systems followed by NODEV are  pseudo file systems, as there is no related device. The */proc/sys* directory contains kernel parameters, some of which are tunable.
* **sysfs** - The device model for the kernel is an object-oriented structure that reflects the devices known by the kernel through the sysfs file system, which is normally mounted on the */sys* directory. When a kernel discovers a new device, it builds an object in the*/sys/devices* directory. The kernel uses a network socket to communicate the new device information to the udevd daemon, which builds an entry in the*/dev* directory. The */sys/fs* directory contains the kernel object structures for media based file systems. The */sys/module* directory contains objects for each loaded kernel module.
* **tmpfs** - The tmpfs file system is often mounted on the*/dev* directory. Since it is a pseudo file system, any data in the */dev*directory is lost when the device is rebooted.
  + Multics
    - Multics was the first operating system to introduce a true [hierarchical storage system](http://multicians.org/features.html#tag11), in which a [directory](http://multicians.org/mgd.html#directory) could contain other directories.
    - For ease of understanding, the file structure may be thought of as a tree of files, some of which are directories. That is, with one exception, each file (e.g., each directory) finds itself directly pointed to by exactly one branch in exactly one directory. The exception is the root directory, or **root**, at the root of the tree. Although it is not explicitly pointed to from any directory, the root is implicitly pointed to by a fictitious branch which is known to the file system.
    - The set of permissions with which a given user may access a particular branch is called the **mode** of the branch for that user. Associated with each branch is an **access control list**, which contains the list of users (or sets of users) along with the corresponding mode associated with each user.
* Power Management
  + Windows
    - The Windows operating system uses power-management hardware to put the computer into a low-power sleep state instead of shutting down completely, so that the system can quickly resume working. The operating system will automatically enter the sleep state when the computer is idle or when the user presses a button to indicate that the current work session is over. To the user, the system appears to be off. While in the sleep state, the computer's processor is not executing code and no work is being accomplished for the user. However, events in the system from both hardware devices and the real-time clock can be enabled to cause the system to exit the sleep state (that is, "wake up") and quickly return to the working state.
    - When the computer is in the sleep state, the computer hardware, the system, and applications running on the computer must be capable of responding immediately to the power switch, communications events, and other actions. If all applications handle power state transitions gracefully, the user will perceive a more elegant and integrated system. Applications that do not handle these transitions can fail when the power is turned off and then on, because of data loss or a dependency on a device that may have been removed.
  + Mac OS X and iOS
    - Has a “power plane” stored in it’s I/O registry to model power dependencies of a computer’s components.
    - Each driver that supports power management is represented as a node in the power plane connected to the parent nodes, which represents devices that provide it power, and with children nodes, which represent devices that it provides power to. The parent of a device in the power plane is typically the driver’s provider class (that is, the same object that the driver is connected to in the service plane of the I/O Registry), although this does not need to be the case.
    - The system’s power management, including transitions from one power state to another, is handled by the I/O Kit. Power management is performed by drivers that run in the kernel; user space drivers that can be written for hardware, such as USB devices, cannot play a part in the power management of that device. The I/O Kit provides support for power management in the IOService superclass from which all Mac OS X drivers are ultimately derived. This makes power management accessible to all drivers, providing that the driver has chosen to insert itself into the power plane.
      * Page 207 Mac OS X and iOS Internals
  + Android
    - Wake locks on Android allow the system to go in to a deeper sleep mode, without being tied to an explicit user action like turning the screen off. The default state of the system with wake locks is that the device is asleep. When the device is running, to keep it from going back to sleep something needs to be holding a wake lock.
    - While the screen is on, the system always holds a wake lock that prevents the device from going to sleep, so it will stay running, as we expect.
    - When the screen is off, however, the system itself does not generally hold a wake lock, so it will stay out of sleep only as long as something else is holding one. When no more wake locks are held, the system goes to sleep, and it can come out of sleep only due to a hardware interrupt.
    - Once the system has gone to sleep, a hardware interrupt will wake it up again, as in a traditional operating system. Some sources of such an interrupt are timebased alarms, events from the cellular radio (such as for an incoming call), incoming network traffic, and presses on certain hardware buttons (such as the power button). Interrupt handlers for these events require one change from standard Linux: they need to aquire an initial wake lock to keep the system running after it handles the interrupt.
    - The wake lock acquired by an interrupt handler must be held long enough to transfer control up the stack to the driver in the kernel that will continue processing the event. That kernel driver is then responsible for acquiring its own wake lock, after which the interrupt wake lock can be safely released without risk of the system going back to sleep.
    - If the driver is then going to deliver this event up to user space, a similar handshake is needed. The driver must ensure that it continues to hold the wake lock until it has delivered the event to a waiting user process and ensured there has been an opportunity there to acquire its own wake lock. This flow may continue across subsystems in user space as well; as long as something is holding a wake lock, we continue performing the desired processing to respond to the event. Once no more wake locks are held, however, the entire system falls back to sleep and all processing stops.
  + Ubuntu
    - UPower is an abstraction for enumerating power devices, listening to device events and querying power history and statistics. Any application or service on the system can access the org.freedesktop.UPower service via the system message bus.
    - UPower also does auto-s3|s4|s5 on critical low battery events.
    - ACPI
      * is a standard for computer hardware that allows the operating system to discover and configure power management for hardware devices. There are three ACPI states that matter:

1. **S1**: All processor caches are flushed, and the CPU(s) stop executing instructions. Power to the CPU(s) and RAM is maintained; devices that do not indicate they must remain on may be powered down.
2. **S2**: The CPU is powered off (unsupported)
3. **S3**: Commonly referred to as Standby, Sleep, or Suspend to RAM. RAM is still powered.
4. **S4**: Hibernate. All content of RAM is saved to the hard drive and the machine is entirely powered down.

Often, you can configure in the BIOS if you want your computer to sleep using S1 or S3. S3 uses less power, but S1 is generally more compatible.

* + - **gnome-power-manager**
      * The gnome-power-manager is a program with a graphical user interface that subscribes itself to power events and acts on them. It shows you the battery status on laptops and dims down the screen if on battery for example. It will also shut down or hibernate the computer after some idle time or before the battery runs out, if a user is logged in.
* Inter-process Communications
  + Multics
    - Inter-[Process](http://multicians.org/mgp.html#process) Communication. The facility for interprocess queueing and transmission of [wakeups](http://multicians.org/mgb.html#block) and associated 72-bit data through a system queue.
    - This development was made possibl e by the availability, in the Multics system, o f certain other capabilities, namely the ability to freely share data bases among (or protect them from) different users, the ability for several o f them to access such a shared data base by refer - ring to it under a single symbolic name 6 , the ability to achieve mutual exclusion among competing computations and the availability of efficien t processor multiplexing capabilities .
    - Data exchange must take place within a shared database.
    - By design, sharing a data base (for whatever reason) presents no difficulty in the Multics system . After agreeing on the segment to be shared fo r storage and retrieval of messages each of the co - operating processes is free to reference the segment by its distinct file system name . Such a reference causes the same segment (same physica l copy in primary memory) to be accessed by each process b .
  + Android
    - The Binder architecture is divided into three layers, shown in Fig. 10-42. At the bottom of the stack is a kernel module that implements the actual cross-process interaction and exposes it through the kernel’s ioctl function. (ioctl is a general-purpose kernel call for sending custom commands to kernel drivers and modules.) On top of the kernel module is a basic object-oriented user-space API, allowing applications to create and interact with IPC endpoints through the IBinder and Binder classes. At the top is an interface-based programming model where applications declare their IPC interfaces and do not otherwise need to worry about the details of how IPC happens in the lower layers.
    - The Binder IPC model is different enough from traditional Linux mechanisms that it cannot be efficiently implemented on top of them purely in user space. In addition, Android does not support most of the System V primitives for cross-process interaction (semaphores, shared memory segments, message queues) because they do not provide robust semantics for cleaning up their resources from buggy or malicious applications.
    - The basic IPC model Binder uses is the RPC (remote procedure call). That is, the sending process is submitting a complete IPC operation to the kernel, which is executed in the receiving process; the sender may block while the receiver executes, allowing a result to be returned back from the call. (Senders optionally may specify they should not block, continuing their execution in parallel with the receiver.) Binder IPC is thus message based, like System V message queues, rather than stream based as in Linux pipes. A message in Binder is referred to as a transaction, and at a higher level can be viewed as a function call across processes
    - Pg. 816 Modern Operating Systems
  + Windows
    - Threads can communicate in a wide variety of ways, including pipes, named pipes, mailslots, sockets, remote procedure calls, and shared files. Pipes have two modes: byte and message, selected at creation time. Byte-mode pipes work the same way as in UNIX. Message-mode pipes are somewhat similar but preserve message boundaries, so that four writes of 128 bytes will be read as four 128-byte messages, and not as one 512-byte message, as might happen with byte-mode pipes. Named pipes also exist and have the same two modes as regular pipes. Named pipes can also be used over a network but regular pipes cannot.
    - Mailslots are a feature of the now-defunct OS/2 operating system implemented in Windows for compatibility. They are similar to pipes in some ways, but not all. For one thing, they are one way, whereas pipes are two way. They could be used over a network but do not provide guaranteed delivery. Finally, they allow the sending process to broadcast a message to many receivers, instead of to just one receiver. Both mailslots and named pipes are implemented as file systems in Windows, rather than executive functions. This allows them to be accessed over the network using the existing remote file-system protocols.
    - Sockets are like pipes, except that they normally connect processes on different machines. For example, one process writes to a socket and another one on a remote machine reads from it. Sockets can also be used to connect processes on the same machine, but since they entail more overhead than pipes, they are generally only used in a networking context. Sockets were originally designed for Berkeley UNIX, and the implementation was made widely available. Some of the Berkeley code and data structures are still present in Windows today, as acknowledged in the release notes for the system.
    - RPCs are a way for process A to have process B call a procedure in B’s address space on A’s behalf and return the result to A. Various restrictions on the parameters exist. For example, it makes no sense to pass a pointer to a different process, so data structures have to be packaged up and transmitted in a nonprocess-specific way. RPC is normally implemented as an abstraction layer on top of a transport layer. In the case of Windows, the transport can be TCP/IP sockets, named pipes, or ALPC. ALPC (Advanced Local Procedure Call) is a message-passing facility in the kernel-mode executive. It is optimized for communicating between processes on the local machine and does not operate across the network. The basic design is for sending messages that generate replies, implementing a lightweight version of remote procedure call which the RPC package can build on top of to provide a richer set of features than available in ALPC. ALPC is implemented using a combination of copying parameters and temporary allocation of shared memory, based on the size of the messages.
    - Finally, processes can share objects. This includes section objects, which can be mapped into the virtual address space of different processes at the same time. All writes done by one process then appear in the address spaces of the other processes. Using this mechanism, the shared buffer used in producer-consumer problems can easily be implemented
    - Processes can also use various types of synchronization objects. Just as Windows provides numerous interprocess communication mechanisms, it also provides numerous synchronization mechanisms, including semaphores, mutexes, critical regions, and events. All of these mechanisms work with threads, not processes, so that when a thread blocks on a semaphore, other threads in that process (if any) are not affected and can continue to run
    - Pg 916 - 917 MOS
  + Mac OS X
    - As an OS X kernel programmer, you have many choices of synchronization mechanisms at your disposal. The kernel itself provides two such mechanisms: locks and semaphores.
    - A lock is used for basic protection of shared resources. Multiple threads can attempt to acquire a lock, but only one thread can actually hold it at any given time (at least for traditional locks—more on this later). While that thread holds the lock, the other threads must wait. There are several different types of locks, differing mainly in what threads do while waiting to acquire them.
    - A semaphore is much like a lock, except that a finite number of threads can hold it simultaneously. Semaphores can be thought of as being much like piles of tokens. Multiple threads can take these tokens, but when there are none left, a thread must wait until another thread returns one. It is important to note that semaphores can be implemented in many different ways, so Mach semaphores may not behave in the same way as semaphores on other platforms.
  + OS 360
    - Definite rules are stated for the ATTACH macro-instruction, telling what parameters can be passed, and how. Techniques exist, however, which a programmer could use to bypass the supervisor's control and establish direct task-to-task intercommunication. It is strongly recommended that only the standard methods be used, because a nonstandard approach can cause system side effects that may be hard to find and to correct, particularly if there is any change in the supervisor due to new equipment, addition of selectable modules, or program updating.
      * IBM Operating System/360 Concepts and Facilities P. 78
* I/O
  + Multics
    - An I/O system has been implemented in the Multics system that facilitates dynamic switching of I/O devices. This switching is accomplished by providing a general interface for all I/O devices that allows all equivalent operations on different devices to be expressed in the same way. Also particular devices are referenced by symbolic names and the binding of names to devices can be dynamically modified. Available I/O operations range from a set of basic I/O calls that require almost no knowledge of the I/O System or the I/O device being used to fully general calls that permit one to take full advantage of all features of an I/O device but require considerable knowledge of the I/O System and the device. The I/O System is described and some popular applications of it, illustrating these features, are presented.
    - In many early operating system designs the software known as the input/output control system (IOCS) played a central conceptual and functional role
    - This is possible partly because two operations sometimes associated with the IOCS have been separated into separate functional units which are made use of by other parts of the system as well as the IOCS. First, the file system [and] Secondly, the traffic controller
  + Unix
    - The I/O subsystem allows a process to communicate with peripheral devices such as disks, tape drives, terminals, printers, and networks, and the kernel modules that control devices are known as device drivers. There is usually a one-to-one correspondence between device drivers and device types: Systems may contain one disk driver to control all disk drives, one terminal driver to control all terminals, and one tape driver to control all tape drives. Installations that have devices from more than one manufacturer — for example, two brands of tape drives — may treat the devices as two different device types and have two separate drivers, because such devices may require different command sequences to operate properly. A device driver controls many physical devices of a given type. For example, one terminal driver may control all terminals connected to the system. The driver distinguishes among the many devices it controls: Output intended for one terminal must not be sent to another.
    - The system supports "software devices," which have no associated physical device
    - The UNIX system contains two types of devices, block devices and raw or character devices. As defined in Chapter 2, block devices, such as disks and tapes, look like random access storage devices to the rest of the system; character devices include all other devices such as terminals and network media. Block devices may have a character device interface, too
      * P 948 Unix Design
  + Windows
    - The I/O manager provides the framework for implementing I/O device drivers and provides a number of executive services specific to configuring, accessing, and performing operations on devices. In Windows, device drivers not only manage physical devices but they also provide extensibility to the operating system. Many functions that are compiled into the kernel on other systems are dynamically loaded and linked by the kernel on Windows, including network protocol stacks and file systems.
      * Pg. 888 Modern Operating Systems
    - Recent versions of Windows have a lot more support for running device drivers in user mode, and this is the preferred model for new device drivers. There are hundreds of thousands of different device drivers for Windows working with more than a million distinct devices. This represents a lot of code to get correct. It is much better if bugs cause a device to become inaccessible by crashing in a usermode process rather than causing the system to crash. Bugs in kernel-mode device drivers are the major source of the dreaded BSOD (Blue Screen Of Death) where Windows detects a fatal error within kernel mode and shuts down or reboots the system. BSOD’s are comparable to kernel panics on UNIX systems.
    - The I/O manager also includes the plug-and-play and device power-management facilities. Plug-and-play comes into action when new devices are detected on the system. The plug-and-play subcomponent is first notified. It works with a service, the user-mode plug-and-play manager, to find the appropriate device driver and load it into the system. Getting the right one is not always easy and sometimes depends on sophisticated matching of the specific hardware device version to a particular version of the drivers. Sometimes a single device supports a standard interface which is supported by multiple different drivers, written by different companies
    - Pg 944 MOS
    - The Windows I/O system consists of the plug-and-play services, the device power manager, the I/O manager, and the device-driver model.
  + Unix
    - The UNIX system provides a primitive form of interprocess communication for tracing processes, useful for debugging. A debugger process, such as sdb, spawns a process to be traced and controls its execution with the ptrace system call, setting and clearing break points, and reading and writing data in its virtual address space, Process tracing thus consists of synchronization of the debugger process and the traced process and controlling the execution of the traced process.
      * Pg 356 Unix Design
    - The UNIX System V 1PC package consists of three mechanisms. Messages allow processes to send formatted data streams to arbitrary processes, shared memory allows processes to share parts of their virtual address space, and semaphores allow processes to synchronize execution. Implemented as a unit, they share common properties.
      * Pg 359 Unix Design
  + Mac OS X
    - In creating OS X, Apple has completely redesigned the Macintosh I/O architecture, providing a framework for simplified driver development that supports many categories of devices. This framework is called the I/O Kit.
    - By starting with properly designed base classes, you gain a head start in writing a new driver; with much of the driver code already written, you need only to fill in the specific code that makes your driver different.
    - n general, all hardware support is provided directly by I/O Kit entities. One exception to this rule is imaging devices such as printers, scanners, and digital cameras (although these do make some use of I/O Kit functionality).
    - Families
      * A family defines a collection of high-level abstractions common to all devices of a particular category that takes the form of C code and C++ classes. Families may include headers, libraries, sample code, test harnesses, and documentation. They provide the API, generic support code, and at least one example driver (in the documentation).
      * Families provide services for many different categories of devices. For example, there are protocol families (such as SCSI, USB, and FireWire), storage families (disk), network families, and families to describe human interface devices (mouse and keyboard). When devices have features in common, the software that supports those features is most likely found in a family.
* Threads
  + Windows
    - Threads are the kernel’s abstraction for scheduling the CPU in Windows. Priorities are assigned to each thread based on the priority value in the containing process. Threads can also be affinitized to run only on certain processors. This helps concurrent programs running on multicore chips or multiprocessors to explicitly spread out work. Each thread has two separate call stacks, one for execution in user mode and one for kernel mode. There is also a TEB (Thread Environment Block) that keeps user-mode data specific to the thread, including per-thread storage (Thread Local Storage) and fields for Win32, language and cultural localization, and other specialized fields that have been added by various facilities.
      * P. 908 Modern Operating System
    - The Win32 thread pool is a facility that builds on top of the Windows thread model to provide a better abstraction for certain types of programs. Thread creation is too expensive to be inv oked every time a program wants to execute a small task concurrently with other tasks in order to take advantage of multiple processors. Tasks can be grouped together into larger tasks but this reduces the amount of exploitable concurrency in the program. An alternative approach is for a program to allocate a limited number of threads, and maintain a queue of tasks that need to be run. As a thread finishes the execution of a task, it takes another one from the queue. This model separates the resource-management issues (how many processors are available and how many threads should be created) from the programming model (what is a task and how are tasks synchronized). Windows formalizes this solution into the Win32 thread pool, a set of APIs for automatically managing a dynamic pool of threads and dispatching tasks to them.
      * P 911 MOS
    - What programmers see as a single Windows thread is actually two threads: one that runs in kernel mode and one that runs in user mode. This is precisely the same model that UNIX has. Each of these threads is allocated its own stack and its own memory to save its registers when not running. The two threads appear to be a single thread because they do not run at the same time. The user thread operates as an extension of the kernel thread, running only when the kernel thread switches to it by returning from kernel mode to user mode. When a user thread wants to perform a system call, encounters a page fault, or is preempted, the system enters kernel mode and switches back to the corresponding kernel thread. It is normally not possible to switch between user threads without first switching to the corresponding kernel thread, switching to the new kernel thread, and then switching to its user thread.
      * Pg 911-912 MOS
    - Every process normally starts out with one thread, but new ones can be created dynamically. Threads form the basis of CPU scheduling, as the operating system always selects a thread to run, not a process. Consequently, every thread has a state (ready, running, blocked, etc), whereas processes do not have scheduling states. Threads can be created dynamically by a Win32 call that specifies the address within the enclosing process’ address space at which it is to start running.
    - Every thread has a thread ID, which is taken from the same space as the process IDs, so a single ID can never be in use for both a process and a thread at the same time. Process and thread IDs are multiples of four because they are actually allocated by the executive using a special handle table set aside for allocating IDs. The system is reusing the scalable handle-management facility shown in Figs. 11-16 and 11-17. The handle table does not have references on objects, but does use the pointer field to point at the process or thread so that the lookup of a process or thread by ID is very efficient. FIFO ordering of the list of free handles is turned on for the ID table in recent versions of Windows so that IDs are not immediately reused. The problems with immediate reuse are explored in the problems at the end of this chapter.
    - A thread normally runs in user mode, but when it makes a system call it switches to kernel mode and continues to run as the same thread with the same properties and limits it had in user mode. Each thread has two stacks, one for use when it is in user mode and one for use when it is in kernel mode. Whenever a thread enters the kernel, it switches to the kernel-mode stack. The values of the user-mode registers are saved in a CONTEXT data structure at the base of the kernel-mode stack. Since the only way for a user-mode thread to not be running is for it to enter the kernel, the CONTEXT for a thread always contains its register state when it is not running. The CONTEXT for each thread can be examined and modified from any process with a handle to the thread. Threads normally run using the access token of their containing process, but in certain cases related to client/server computing, a thread running in a service process can impersonate its client, using a temporary access token based on the client’s token so it can perform operations on the client’s behalf. (In general a service cannot use the client’s actual token, as the client and server may be running on different systems.) Threads are also the normal focal point for I/O. Threads block when performing synchronous I/O, and the outstanding I/O request packets for asynchronous I/O are linked to the thread. When a thread is finished executing, it can exit. Any I/O requests pending for the thread will be canceled. When the last thread still active in a process exits, the process terminates. It is important to realize that threads are a scheduling concept, not a resource-ownership concept. Any thread is able to access all the objects that belong to its process. All it has to do is use the handle value and make the appropriate Win32 call. There is no restriction on a thread that it cannot access an object because a different thread created or opened it. The system does not even keep track of which thread created which object. Once an object handle has been put in a process’ handle table, any thread in the process can use it, even it if is impersonating a different user. As described previously, in addition to the normal threads that run within user processes Windows has a number of system threads that run only in kernel mode and are not associated with any user process. All such system threads run in a special process called the system process. This process does not have a user-mode address space. It provides the environment that threads execute in when they are not operating on behalf of a specific user-mode process. We will study some of these threads later when we come to memory management. Some perform administrative tasks, such as writing dirty pages to the disk, while others form the pool of worker threads that are assigned to run specific short-term tasks delegated by executive components or drivers that need to get some work done in the system process.
  + Mac OS X
    - Threads can migrate between priority levels for a number of reasons, largely as an artifact of the time sharing algorithm used. However, this migration is within a given band.
    - Threads marked as being real-time priority are also special in the eyes of the scheduler. A real-time thread tells the scheduler that it needs to run for A cycles out of the next B cycles. For example, it might need to run for 3000 out of the next 7000 clock cycles in order to keep up. It also tells the scheduler whether those cycles must be contiguous. Using long contiguous quanta is generally frowned upon but is occasionally necessary for specialized real-time applications.
  + Android
    - Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority. Each thread may or may not also be marked as a daemon. When code running in some thread creates a new Thread object, the new thread has its priority initially set equal to the priority of the creating thread, and is a daemon thread if and only if the creating thread is a daemon.
    - When a Java Virtual Machine starts up, there is usually a single non-daemon thread (which typically calls the method named main of some designated class). The Java Virtual Machine continues to execute threads until either of the following occurs:
    - The exit method of class Runtime has been called and the security manager has permitted the exit operation to take place.
    - All threads that are not daemon threads have died, either by returning from the call to the run method or by throwing an exception that propagates beyond the run method.
    - -Android Developer Website (Threads)