Plagiarism Checker X Originality Report

Similarity Found: 12%

Date: Sunday, June 07, 2020
Statistics: 627 words Plagiarized / 5324 Total words
Remarks: Low Plagiarism Detected - Your Document needs Optional Improvement.

Overview on algorithms of android concerning basic OS topics Youssef Moh. Sobhy Alexandria, Egypt Abstract Due to the recent advances in the technology, the smart phones replaced the traditional mobile phones, While the Apple iPhone rede?ned the term smart phone during its ?rst two years of release, Google's Android platform for mobile devices has quickly developed into a serious open source alternative. It also developed from it's ?rst phone in October 2008 to being the most popular smart phone op- erating system in the world by 2012 with 11 main versions and 30 API levels.

According to Statista, Android maintained it's position as the world 's leading mobile operating system in late 2019, controlling the mobile OS market with a 74.13 percent share. Android is a Linux based operating system, designed primarily for touch screen mobile devices such as smart phones and tablets. This paper demonstrates year- round android evelotion, explores the android operating system from it's main aspects. It explains android structure, How android manages processes, used CPU scheduling and Virtual memory management algorithms in android. It also high- lights memory management techniques and how android deals with low memory situations.

Keywords: Android, CPU Scheduling, Memory Management 1. Introduction Android is an operating system based on Linux which is designed for touch screen mobile devices such as smart phones and tablets. In the last 13 years, The operating system has changed a lot from a Primitive phones to new smartphones. Android is one of the most commonly used OS smartphone of these days. The an- droid is an operating system formed in October 2003 in Palo Alto , California by Andy Rubin, R.Miner and N.Sears.

Android was initially developed for cameras, Preprint submitted to Journal Name June 7, 2020 but the company then concluded that the cameras market was not big enough for it’s ambitions, and ?ve months later it redirected its resources and marketed An- droid as a smartphone operating system that would compete with Symbian ,IOS and Microsoft Windows Mobile [1] .The platform was created by Android Inc. which was bought by Google and released as the Android Open Source Project (AOSP) in 2007 which is led by Google.

A group of 78 different companies like Samsung, Sony, Intel and many more formed the Open Handset Alliance (OHA) that is dedicated to develop and distribute Android. The software can be downloaded freely from a central repository and license-related modi?cations[2] . Android has seen several updates that have enhanced the operating system in- crementally, each new major release is named after a candy or a sugar treat in alphabetical order.

Google stopped calling its android models after a dessert after it released Android 10[3]. Android has been the best-selling OS worldwide on smartphones since 2011 and on tablets since 2013. As of May 2017, It has over two billion active monthly user, the highest installed base rate of any OS, and as of March 2020, the Google Play Store offers over 3 million apps. The current stable version is Android 10, released on September 3, 2019. Android 11 will be announced at Google I/O online conference in June 3. 2. Basics & Background 2.1. OS structure The operating system is a framework that permits the user application pro- grams to communicate with the hardware of the machine.

Because the operat- ing system is such a complicated device, it should be designed with the utmost caution so that it can be conveniently accessed and changed. An simple way to achieve so is to build a component of the operating system. Each of these parts should be clearly de?ned with clear inputs , outputs and functions. Each OS shall be responsible for Process Control, Primary Memory Management , File Management, I / O Device Management, Secondary Management, Networking, Command-Interpreter Device. 2.2.

PROCESS MANAGEMENT The process is a program in execution: (the program is passive, the process is active). The process has resources (CPU time, ?les) and attributes that need to be handled. Process management includes: • Process Scheduling (priority, time-management, ...) • Creation / termination • Block / Deblock (Suspension / 2 Resumption) • Synchronization • Communication: • deadlock managment • De- bugging 2.3. Memory management Memory management is the operation of an operating system that treats or controls primary memory and transfers tasks between main memory and disk dur- ing operation.

memory management keeps track of each and every memory lo- cation, irrespective of whether it is assigned to a process or free. It checks how much memory is to be assigned to the process. It controls which process will get the memory at what time. It tracks memory statues is freed or unallocated. It is mainly responsible for, but not limited to: • Allocation / de-allocation for method, ?le, I / O. • Maintenance of multiple processes at a time • Keep tracking of memory • Movement of process memory to/from secondary storage. 2.4. FILE MANAGEMENT A ?le is a set of related information as speci?ed by its author.

Typically, ?les Represent programs (source and object forms) and data. The operating system shall be responsible for the following activities in connection with the ?le Man- aging: Creation and deletion of paper. Creation and deletion of directory. to access ?les and folders. Mapping ?les to secondary storage. Backup ?le to secure (non-volatile) storage media. 2.5. I/O MANAGEMENT it's main task are: • Buffer caching system • Generic device driver code • Drivers for each device to read or write requests into disk. commands. 2.6.

Virtual Memory It's operating system (OS) memory management feature that appeared in 1956 due to lack of memory space. By temporarily transferring pages from Random Access Memory (RAM) to disk storage, a computer can compensate for physical memory shortages. This process is performed temporarily and is designed to act as a combination of RAM and hard disk space. By using active RAM memory & inactive memory in hard disk, The space taken by virtual address is increased to form a contiguous addresses. To hold the application with its data. This means that virtual memory can move data from it to a space, called a paging ?le, when RAM is running low. 3 2.7. Paging Paging is an important aspect of the design of the virtual memory of modern operating systems.

It lets programs exceed the available physical memory. In paging, The operating system recovers data from secondary storage in blocks of the same size. 2.8. Memory-mapped A memory-mapped ?le is a virtual memory section that has been allocated a direct byte-byte association with any component of a ?le or a ?le-like resource. This resource is usually a ?le that is physically present on disk, but can also be a computer, shared memory entity, or any resource that can be accessed by the operating system via a ?le descriptor.

Once it's available, The similarity between the ?le and the memory space helps programs to view the mapped section as if it were main memory. 2.9. Multitasking It's the method of making a computer performing several functions concur- rently. During multitasking, functions such as listening to music or sending a message may be done in the background while performing others in the fore- ground. It relays the ?exibility of the operating system, the complexity of the CPU and the pace and memory (RAM) and storage capacities as several activi- ties are conducted concurrently by the Processor through moving between them.

Switches arise so often that users can communicate with each program when it is running. The OS carries out the following multitasking activities The user gives instructions directly to the operating system or program and receives an immediate response. OS can handle multiple operations and can run more than program at a time. Multitasking Operational Systems are also known as Time-Sharing Systems. These operating systems have been developed to ensure the interactive use of the computer system at a reasonable cost. A time-shared system uses multi-programming techniques and CPU schedul- ing to allow each user a small amount of a time-shared CPU.

Every user has at least one separate program in the memory. 4 3. Android structure Linux kernel: the bottom layer is Linux - Linux 3.6 has approximately 120 patches. This provides an abstraction level between device hardware and includes all the essential hardware drivers such as camera, display etc. In addition, the kernel of the linux is very good at networking and a large array of computer drivers that take the pain out of peripheral hardware interfaces. 3.1.

Android Libraries (Native Components) This subsection covers certain Java based libraries that are unique to the devel- opment of Android. Examples of libraries in this subsection include application framework libraries, as well as those that enable user interface creation, design illustration, and database access. The following is a description of some main Android libraries available Android.app: provides access to the App model and is the main component of all Android apps. Android.database: Used to access content providers' published data, and in- cludes SQLite databases Android.os: provides the main operating system services such as system ser- vices ,messages and inter-process communications between applications. Android.opengl: The OpenGL ES 3D graphics Java interface to render the API. Android.widget: components such as buttons, widgets, labels, list views etc. 5 Android.webkit: A set of classes designed to incorporate Web browsing capa- bilities into applications. 3.2.

Android Runtime the third section is about the architecture and it's available on the second layer from the bottom. This section provides a key component called Dalvik Virtual Machine which is nearly similar to Java Virtual Machine (JVM) but it's uniquely designed and optimized for Android. The Dalvik VM uses Linux main features such as multi-threading and memory managementment , which is intrinsic to Java language. The Dalvik VM allows each Android application to run in it's own process, with Dalvik virtual machine as its own case.

Apart from the main one, the Android runtime also provides a set of core li- braries that allow Android application to be written using standard Java program- ming language. 3.3. Application Framework The Application Framework layer provides applications with many higher- level services in Java class format. Those facilities developing apps. Action Manager: Manages all facets of the lifecycle and operation stack of an program. Content Providers: Allows the publication and sharing of data with other ap- plications. Resource Manager: offers access to non-code embedded tools including strings, color preferences, and con?gurations of the user interface.

noti?cation Manager: Allows the user to access updates and feedback from apps. System View: An extensible series of views used to construct user interfaces for applications. 3.4. Applications All applications, including the ones that come with Android OS are written at this level. This means a couple important things: All applications are written in Java, which means they are able to be run on ANY installation of Android OS as .apk. These are hardware independent and are compiled into dex format.

Any software developer can write the same applications as Google. So UI (including the homescreen) can be customized. 6 4. Process Management Any android process can be in one of ?ve different states at any time, It's called Process lifespan Hierarchy, from the most important to the least important: 1. Foreground process: The application you are using is considered to be the process in the foreground. Other processes can also be considered primary pro- cesses — for instance, if they interact with the process currently in the foreground. 2.

Visible process: A visible process is not at the foreground but still affects what you have on your mobile screen. For instance, the process in the foreground may be a dialog that allows you to see an application behind it. 3. Service process: A service process is not linked to any app on your screen that is visible. In the background, though, it does something like playing music or downloading data in the background. 4. Background process: At present, background processes are not available to the user. They have no effect on the phone use experience. Many background processes are currently under way at any given time.

Such processes in the back- ground function as paused apps. They are kept in memory so that you can resume using them quickly when you return to them, but they do not use valuable CPU time. 5. Empty process: it's no longer contains info about an app. used to speed up app launches later, it may be kept around as cache ?les and eventually the system may kill it as needed. 4.1. Applications processes Android process is the same as process Linux. by necessity, Every .apk in- stalled runs in its own Linux method. by default, There is also 1 thread per process. The main thread has a looper instance for handling messages from the message queue and it calls Looper.loop() in its every iteration of run() method.

It's a looper 's job to pop up messages from the message queue and invoke the appropriate methods to handle them. Any process gets initiated whenever necessary. Whenever a user or some other system component requests any component (could be a service, activity or in- tended receiver) that belongs to an apk, if it is not already running, the system spins out a new process for your apk. General processes continue to run until system kills. 4.2. Process Start up Like many of the Linux-based systems, the bootloader ?rstly loads the kernel and immediately begins the init process. it generates low-level processes called 7 "daemons".

These daemons usually handle low-level hardware interfaces, includ- ing the radio interface. Init cycle begins a process called 'Zygote.' this is the beginning of the Android platform. The process that initializes the very ?rst instance of the Dalvik virtual machine and preloads all the common classed used by the application framework and the various applications. Then it begins listening to future requests to create new vms for handling new device processes. After receiving a new request, it forks itself to build a new process to be a pre-initialized vm instance. After zygote, the runtime process starts at init.

the zygote initiate a process called server process. It starts all of the core platform services , e.g. hardware services. At this stage, the system is ready to start the process which displays in the home screen. When a user touches any icon app, the app will be launched and the following things will happen: The click event will be translated into startAc- tivity(intent) call that will be routed to startActivity(intent) call in ActivityMan- agerService via Binder IPC. ActvityManagerService takes a couple of steps- • the ?rst thing is to gather information about the targeted object by resolveIn- tent() in PackageManager object.PackageManager.MATCH DEFAULT ONLY and PackageManager.GET SHARED LIBRARY FILES ?ags are used as a default choice. • The essential information is saved back to the intent object to avoid repeat- ing this step. • check if user has enough privileges to invoke the target component of the intent.

This is done by calling method GrantUriPermissionLocked(). 8 • If the user has suf?cient permissions, ActivityManagerService checks if a new task requires launching of the target activity. The task creation depends on Intent ?ags such as FLAG ACTIVITY NEW TASK and other ?ags such as FLAG ACTIVITY CLEAR TOP. • ?nally,check if the ProcessRecord is existed for the process. If the ProcessRecord is null, the ActivityManager has to create a new process to instantiate the target component. There are three distinct phases of process launch : 1. Process Creation 2. Binding Application 3.

Launching Activity Process Creation : The ActivityManagerService will initiate a new process by invoking the start- ProcessLocked() method that sends arguments to the Zygote process. Zygote forks and calls ZygoteInit.main() which then instantiates the ActivityThread ob- ject as well as reterning the pid of the newly created process. ActivityThread will start the message loop by calling Looper.prepareLoop() and Looper.loop() after- wards. The following describes in detail the call sequence Application Binding : Application Binding: The next step is to add the method to the particular ap- plication.This is done by calling bindApplication, the Thread object.

This method sends BIND APPLICATION message to the queue of messages. This message is collected by the handler object which invokes the handleMessage() method 9 to trigger the speci?c action of the message - handleBindApplication(). This method will invoke makeApplication() method which will load separate app spe- ci?c classes into memory. This call sequence is shown in following ?gure. Activity Launch : After the previous stage, the system containing the process which is responsi- ble for the application with all application classes are loaded in process's private memory.

The call sequence for an operation to start is normal between a newly generated process and an existing one. The actual process of launching starts in realStartActivity() method which calls sheduleLaunchActivity() on the appli- cation thread object. This method sends LAUNCH ACTIVITY message to the message queue. The message is managed by handleLaunchActivity() method as shown below. Assuming that user clicks on Video plugin or browser application. the call sequence will launch the activity as shown in the ?gure. The Activity starts with onCreate() method call. then comes to foreground with onRestart() call and starts interacting with the user with onStart() call. 10 5.

CPU Scheduling Algorithm Android is Linux based, and uses the scheduling mechanisms of the Linux kernel to determine scheduling policies. This applies to Java code and threads too. The Linux schedule strategy combines static and dynamic goals. Processes can be given an initial priority from 19 to -20 (very low to very high priority). This priority will assure that higher priority processes will get more CPU time when when needed. These level are however dynamic, low level priority tasks that do not consume their CPU time will ?ne their dynamic priority increased. This dynamic behaviour results is an overall better responsiveness.

In terms of dynamic priorities it is ensured that lower priority processes will always have a lower dynamic priority than processes with real-time priorities. Android uses two main different CPU scheduling mechanisms to schedule process levels real-time scheduling Linux kernel provides two real-time scheduling algo- rithms which are SCHED FIFO and SCHED RR. The main real-time policy is SCHED FIFO. Once a SCHED FIFO task begins to run it keeps on running until a higher-priority real-time process willingly yields the cpu, blocks or is preempted. There are no timestamps.

All other lesser-priority tasks will not be scheduled until the CPU is aban- doned. Two SCHED FIFO tasks with equivalent priority do not pre-empt one another, diffrence between SCHED RR and SCHED FIFO, that these tasks are as- signed timestamps depending on their urgency and run until they are completed. Use the SCHED NORMAL scheduling policy for non-real - time tasks. The An- droid kernel is designed to allow multi scheduling of real-time operations, the path of ?le system monitoring this process in found under / dev / cpuctl.

Android uses two different scheduling classes (using linux cgroups) bg non interactive and default (foreground). The con?guration is that bg non interactive is low priority and can maximum utilize 5% of the cpu (in- cluding all background tasks) and foreground 95%. Forground means either an Activity or a service that is started foregound. startup Services are always running in bg non interactive unless they have been raised to foreground scheduling using startForeground (applications are always at foreground level). 5.1.

JVM thread and process scheduling The Android system runs a set of unix processes. Some are native processes, but others are processes that run a Java virtual machine. These processes are 11 normally multi-threaded, and all android threads are native pthreads (no green threads). There are two ways that Calling Thread.setPriority can change prior- ities. This is part of the standard Java API and contains a value from MIN PRIORITY(1) to MAX PRIORITY(10). As all threads are pthreads these pri- orities will be mapped to unix process priorities (MIN PRIORITY being 19 and MAX PRIORITY -8). Thread.priority Java name Android property name Unix priority 1 MIN PRIORITY ANDROID PRIORITY LOWEST 19 2 ANDROID PRIORITY BACKGROUND + 6 16 3 ANDROID PRIORITY BACKGROUND + 3 13 4 ANDROID PRIORITY BACKGROUND 10 5 NORM PRIORITY ANDROID PRIORITY NORMAL 0 6 ANDROID PRIORITY NORMAL - 2 -2 7 ANDROID PRIORITY NORMAL - 4 -4 8 ANDROID PRIORITY URGENT DISPLAY + 3 -5 9 ANDROID PRIORITY URGENT DISPLAY + 2 -6 10 MAX PRIORITY ANDROID PRIORITY URGENT DISPLAY -8 The second way to set priorities is to call android.os.Process.setThreadPriority().

This allows to set the pririty to higer priorities for that Declare: in your Android- Manifest and call Process.setThreadPriority(Process.myTid(), Process.THREAD PRIORITY URGENT

DISPLAY) frameworks/base/include/utils/threads.h ANDROID PRIORITY LOWEST = 20, /* use for background tasks */ ANDROID PRIORITY BACKGROUND = 12, /* most threads run at normal priority */ ANDROID PRIORITY NORMAL = 0, /* threads currently running a UI that the user is interacting with */ ANDROID PRIORITY FOREGROUND = -2, /* the main UI thread has a slightly more favorable priority */ ANDROID PRIORITY DISPLAY = -4, 12 /* ui service treads might want to run at a urgent display (uncommon) */ ANDROID PRIORITY URGENT DISPLAY = -8, /* all normal audio threads */ ANDROID PRIORITY AUDIO = -14, /* service audio threads (uncommon) */ ANDROID PRIORITY URGENT AUDIO = -19, /* regular process not allowed to use this level */ ANDROID PRIORITY HIGHEST = -20, 6.

Memory Management The Android architecture runs on the idea that any free memory is wasted(unused) memory. It's just struggling to ?ll all of the available memory. The system keeps all apps in the memory after they've been closed so the user can switch back to the closed apps at any time. Android devices often run with very little free memory, for this reason. Memory management is essential for proper allocation of memory between important system processes and multiple user applications. The fundamentals of how Android allocates memory for the device and user applications are covered in the following section.

It also describes how the oper- ating system responds to circumstances with low memory situations. 6.1. Memory pages RAM is fractured into pages. Every page is usually 4 KB of memory. Pages are either considered safe, or licensed. Unused RAM is considered as a Free Pages. The pages used are RAMs that the program regularly uses, which are divided into the following categories: • Cached: Memory backed by a ?le on storage (for example, code or memory- mapped ?les). There are two types of cached memory – Private: Owned by one process and not shared * Clean: Unmodi?ed copy of a ?le may be removed to maximize free memory 13 * Dirty: Modi?ed copy of the ?le to storage; can be moved to or compressed to zRAM to increase free memory – Shared: Used by multiple processes * Clean: Unmodi?ed copy of the ?le can be removed to increase free memory. * Dirty:Changed storage copy of the ?le; allows changes to be writ- ten back to the storage ?le to boost free memory.

• Anonymous: Memory not backed by a ?le on storage – Dirty: Can be moved/compressed in zRAM to increase free memory Clean Pages contain an same copy of a stored ?le (or part of a ?le). A clean page becomes a dirty page because it no longer contains an exact copy of the ?le like,the product of an application process . Clean pages can be removed because the data from the database can still be regenerated; dirty pages can not be removed, or data will be lost. Within time, the proportions of free and used pages differ as the program actively manages RAM.

The concepts presented in this section are key for the management of low-memory situations. Those are described in greater depth in the next section 6.2. low memory management Android has two main techniques to counter low memory situations: the kernel swap daemon and low-memory killer. 6.2.1. kernel swap daemon The kernel swap daemon also known as (kswapd) is a main part of the kernel and transform the memory used to free memory. The kswapd will become active when the free memory of the phone is running low.

Once free memory slips below the small threshold, kswapd begins to restore memory. When free memory reaches a high threshold, kswapd stops retrieving memory. kernel swap daemon can retrieve clean pages because they are backed up in storage without modi?cation. If a process attempts to retrieve a deleted clean page, the kernal will copy the page from main storage to the RAM. This method is referred to as demand paging. kernel swap daemon can redirect cached and anonymous private dirty pages into zRAM,(place of compression).

This frees up the available memory in the 14 Figure 1: Clean page, backed by storage, deleted Figure 2: Dirty page moved to zRAM and compressed RAM. When the process attempts to move a dirty page in the zRAM, the page will be uncompressed and will return to RAM. If the process related with a compressed page is killed, the page also will be deleted from zRAM. the system may start killing processes. If the amount of free memory went down to a serious level. 6.2.2. Low-memory killer if Kswapd has not been able to free enough memory for the system.

In this case, the system uses TrimMemory() to tell the app that the memory is at low level and that it should minimize it’s memory allocation. If it’s not enough, the kernel will use low-memory killer (LMK) to start killing processes(considering priority) to free the memory. LMK uses a score called ”out of memory” to prioritize run- ning processes to decide the processes that will be kill. Background processes are killed ?rst, and system processes killed last. 6.3. Garbage collection The managed memory environment, such as the Dalvik virtual machine or ART (after android 5.0), it keeps track of all memory allocations.

When it says that the program is no longer in the memory, it will be released to the heap, without any interference from the programmer. The method for restoring unused memory in a controlled memory is well known as garbage collection. Garbage collecting 15 has two objectives: to identify the objects in the memory that can not be retrieved in the future and to recover the resources used by such objects. the heap of the Android’s memory is a generational one, meaning that there are various seals of allocations that it records, depending on the projected existence and scale of the object being allocated.

Newly allocated objects, for example, belong to the Young Generation. If the object stays alive long enough, it may be transferred to an older generation, and then transferred to a permanent generation. That generation of heaps has its limit. Whenever a generation starts ?lling up, the machine will execute the garbage collection to free the memory. The period of the garbage collection depends on the group of objects it gathers and how many active objects there are in each generation. The system has a set of criteria in place to determine when to excute garbage collector. When the conditions are met, the system will stop executing the process and starts garbage collection.

If garbage collection happens in the midst of an intense processing cycle, such as an playing games or during music playback, the processing time can be increased. 6.3.1. Share memory Android is always trying to share available RAM across all available processes in order for everything to ?t in ram. This can be done in the next ways: Each new application process is forked by an existing process called Zygote. The Zygote cycle begins when the machine starts up and loads the main frame- works and tools. To start any new process, the system begins to forks Zygote process, it runs the app code in the new process.

This technique permits most of the RAM pages to be assigned to the framework code also it allows the available resources to be shared all across app processes. Most of the static data is mmapped in a process. This method allows all re- quired data to be exchanged between systems, and allows it to be paged out as appropriate. 16 7. Virtual memory Android Runtime (ART) -introduced to android 4.4 as a beta feature and be- came ?nal in android 5.0 - and Dalvik virtual machines use paging and memory mapping to manage memory. The main visible bene?t of this strategy is that pro- grams can be greater than physical memory and run normally. Virtual memory is usually for two main purposes.

First, it allows the physical memory to be extended by the disk. Second , it protects memory, because every virtual memory address will be translated to a physical address. The following situations arise when the whole program is not needed to be fully loaded into the main memory. • User written error handling routines are only used when a data or computa- tion error has occurred. • Such software functions may not used frequently. • A ?xed amount of address space is assigned to many tables even an only small amount of the table is used. So, the ability to execute a program that uses partial amount of memory would counterbalanced many of the bene?ts.

• lesser amount of I / Os is needed to load or swap the memory of each user program.

7.1. Page Replacement Algorithm Page Replacement Algorithms are the methods used by the Operating System to determine which memory pages to move or write to the disk when any memory page has to be allocated. Paging occurs whenever a page error happens and a free memory page can not be used for the purpose of allocation accounting. it occurs due to that pages are not available or that the number of free pages is less than the required pages.

If the page that has been chosen for replacement and has been paged out is referenced again, it needs to be read from the disk, and that involves I / O comple- tion. This method determines the ef?ciency of any page replacement algorithm: the longer time you wait for the page-ins, the faster the algorithm. A page replacement algorithm looks at the minimal details on accessing the pages given by the hardware and tries to choose which pages to be replaced to reduce the overall amount of pages lost while matching them with the costs of primary storage and processing time of the algorithm itself. There are a number of common page replacement algorithms. Some of them are explained in the following subsections. 17 7.2.

First In First Out (FIFO) algorithm The oldest page in the main memory is the one that will be picked for replacment. it's Easy to implement, new pages are added to the head and page replacments occurs from the tail 7.3. Optimal Page algorithm this algorithm has the lowest page-fault rate of all algorithms. it's also called OPT or MIN. it depend on Replacing the page that will not be used for the longest amount of time. Use the time when a page is not used. 18 7.4. Least Recently Used any Page which hasn't been used for the longest period of time in main mem- ory will be selected for replacement. 8.

Conclusion Android is a much more ?exible operating system than iOS it's truly open, free development platform based on linux and open source. Android has grown rapidly over the years to become the most widely used smartphone operating system in the world. It's because Android doesn't release 1 phone from 1 company with 1 new OS a year, but countless phones from a variety of companies, all year round, are slowly evolving day-by-day. Android's ability to customize is unparalleled relative to Apple's apps, enabling users to modify and con?gure nearly any part of Android that other iPhone users will never conceive about.

Android is exceptional and incomparable to many smartphone operating systems.

INTERNET SOURCES:

<1% -
https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operati

ng-systems-since-2009/
1% - https://en.wikipedia.org/wiki/Android_(operating_system)
<1% - http://uzgx.notaioaosta.it/pubg-airtel-trick.html
<1% -
https://books.google.com.eg/books?id=lR7SBQAAQBAJ&pg=PA11&lpg=PA11&dq=whi
ch+was+bought+by+Google+and+released+as+the+Android+Open+Source+Project+
(AOSP)+in+2007+which+is+led+by+Google.&source=bl&ots=E3r5pndSn3&sig=ACfU3
U2KZNffyGGe1EvLxkHrnhe8OoCBoQ&hl=ar&sa=X&ved=2ahUKEwjJ4Imb_N7pAhUySBU
IHad-CD0Q6AEwCnoECAsQAQ
<1% -
https://ar.wikipedia.org/wiki/%D8%A7%D9%84%D8%A7%D8%AA%D8%AD%D8%A7%D
8%AF_%D8%A7%D9%84%D9%85%D9%81%D8%AA%D9%88%D8%AD_%D9%84%D9%8
4%D9%87%D9%88%D8%A7%D8%AA%D9%81_%D8%A7%D9%84%D9%86%D9%82%D
8%A7%D9%84%D8%A9
<1% -
https://www.google.com/search?num=15&as_qdr=all&q=Android+has+seen+several+
updates+that+have+enhanced+the+operating+system+incrementally,+each+new+maj
or+release+is+named+after+a+candy+or+a+sugar+treat+in+alphabetical+order.&spe
ll=1&sa=X&ved=2ahUKEwiktKrtuO7pAhXBXSsKHTKaDi0QBSgAegQIERAt
<1% - https://en.wikipedia.org/wiki/Process_management_(computing)
1% - https://www.tutorialspoint.com/operating_system/os_memory_management.htm
<1% - https://en.wikipedia.org/wiki/Memory_management_(operating_systems)
<1% - https://www.cs.rutgers.edu/~pxk/416/notes/11-devices.html
<1% - https://g.elite-bamboo.ru/2182
<1% - https://searchstorage.techtarget.com/definition/RAM-random-access-memory
<1% - https://en.wikipedia.org/wiki/Paging
<1% - http://www.ebookbou.edu.bd/Books/Text/SST/DCSA/DCA_2302_full.pdf
<1% -
https://www.researchgate.net/publication/23526710_Introducing_Multitasking_to_the_St
udy_of_Travel_and_ICT_Examining_its_Extent_and_Assessing_its_Potential_Importance
<1% -
https://books.google.com.eg/books?id=sJ-dDwAAQBAJ&pg=PA180&lpg=PA180&dq=
OS+can+handle+multiple+operations+and+can+run+more+than+program+at+a+tim
e.+Multitasking+Operational+Systems+are+also+known+as+Time-Sharing+Systems.&
source=bl&ots=1a-SsNMcKG&sig=ACfU3U39zYzjPd-LxNmDiN0hrjtnTK8G2w&hl=ar&sa
=X&ved=2ahUKEwjE6J3kue7pAhVXWH0KHUDkCPMQ6AEwAHoECAoQAQ
<1% - https://www.tutorialspoint.com/operating_system/os_properties.htm
1% - https://www.tutorialspoint.com/android/android_architecture.htm
<1% -
https://books.google.com.eg/books?id=3NFjDwAAQBAJ&pg=PA27&lpg=PA27&dq=Thi

s+provides+an+abstraction+level+between+device+hardware+and+includes+all+the+essential+hardware+drivers+such+as+camera,+display+etc.&source=bl&ots=56ry08bp85&sig=ACfU3U0xajsLK9YWQMBKXB7sW2ncifwJPg&hl=ar&sa=X&ved=2ahUKEwiwyvqOudfpAhVMyhoKHWEwALoQ6AEwC3oECA8QAQ
<1% -
https://www.techotopia.com/index.php/An_Overview_of_the_Android_Architecture_(Android_Studio)
<1% - https://www.coursehero.com/file/23015143/Android/
<1% -
https://books.google.com.eg/books?id=tkZFDwAAQBAJ&pg=PA7&lpg=PA7&dq=Android+Runtime+the+third+section+is+about+the+architecture+and+it+s+available+on+the+second+layer+from+the+bottom.&source=bl&ots=LzJmbUMjlA&sig=ACfU3U3BA-HhgS7oIvVbkLjQRhUzsDdNLA&hl=ar&sa=X&ved=2ahUKEwjH2vqHuu7pAhXHe30KHS-KBCkQ6AEwDHoECA0QAQ
<1% -
https://www.sciencedirect.com/topics/computer-science/java-programming-language
<1% - http://www.gpcet.ac.in/wp-content/uploads/2018/08/mad-notes-unit-1.pdf
<1% - http://lim.univ-reunion.fr/staff/fred/Doc/Dalvik/Analysis-of-Dalvik-VM.pdf
<1% -
https://books.google.com.eg/books?id=fpjlDAAAQBAJ&pg=PA605&lpg=PA605&dq=Applications+All+applications,+including+the+ones+that+come+with+Android+OS+are+written+at+this+level.&source=bl&ots=Gb3tnnMk31&sig=ACfU3U0Y2v6PydWoay4GnkW60Ba8YcZ0Wg&hl=ar&sa=X&ved=2ahUKEwibhZyjsNTpAhWMoBQKHRWoCPYQ6AEwAHoECAwQAQ
<1% - http://www.cs.umd.edu/projects/PL/redexer/tutorial.html
1% -
https://www.howtogeek.com/161225/htg-explains-how-android-manages-processes/
<1% - https://dzone.com/articles/android-internals-the-android-os-bootup-process
<1% - http://multi-core-dump.blogspot.com/2010/04/
1% -
https://android.jlelse.eu/android-application-launch-explained-from-zygote-to-your-activity-oncreate-8a8f036864b
<1% - https://link.springer.com/content/pdf/10.1007%2F978-981-13-1026-3.pdf
<1% -
http://tonersytecnologia.com/inolsk41kfjhgld/tobpaow25kfd.php?wehjcv2y5gfj=launching-a-wpf-window-in-a-separate-thread-part-2
<1% - http://multi-core-dump.blogspot.com/2010/
<1% - https://anandgharu.files.wordpress.com/2018/02/spos-assignment-no-13.pdf
<1% - https://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html
<1% - https://www.linuxjournal.com/article/10165

<1% -
https://medium.com/hungys-blog/linux-kernel-process-scheduling-8ce05939fabd
<1% -
https://medium.com/@nimesh.ekanayaka7/android-operating-system-277623c867d1
<1% -
https://mafiadoc.com/download/android-operating-system_5ba71a15097c471e168b4695.html
<1% - https://www.andreasschrade.com/2017/03/28/android-threading/
2% - https://developer.android.com/topic/performance/memory-management
<1% - http://www.who.int/water_sanitation_health/dwq/gdwqvol32ed.pdf?ua=1
<1% -
https://developer.mozilla.org/en-US/docs/Archive/B2G_OS/Platform/Out_of_memory_management_on_Firefox_OS
1% - https://developer.android.com/topic/performance/memory-overview
<1% - https://www.tutorialspoint.com/operating_system/os_virtual_memory.htm