MARTÍN PÉREZ

CONDITIONAL RESOURCE MANAGEMENT FOR MOBILE DEVICES

This dissertation submitted to the Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná in partial fulfilment of the requirements for the Degree of Masters of Applied Computing

Curitiba, PR - Brazil
August, 2014

MARTÍN PÉREZ

CONDITIONAL RESOURCE MANAGEMENT FOR MOBILE DEVICES

This dissertation submitted to the Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná in partial fulfilment of the requirements for the Degree of Masters of Applied Computing

Field: *Computer Systems Engineering*

Advisor: Prof. Dr. Carlos Alberto Maziero

Curitiba, PR - Brazil
August, 2014

Pérez, Martín.

GERENCIAMENTO DE RECURSOS CONDICIONAL PARA DISPOSITIVOS MÓVEIS. Curitiba, 2014. 57p.

Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Computação Aplicada.

1. Dispositivos Móveis 2. Gerenciamento de Recursos 3. Desempenho. I. Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Computação Aplicada II-t.

Acknowledgements

I would like to thank all professors from the PPGCA and CPGEI programs from all learning experiences that I had during this journey. Special thanks Prof. Dr. Carlos Maziero, my advisor, that helped me to broaden my knowledge about Operating Systems in general, guided me during times of uncertainty, figured out a research opportunities in this area and played a key in my masters, being available to shed some light through difficult times.

I am also very thankful Prof. Dr. André Schneider de Oliveira, Prof. Dr. . Luis C. E. de Bona, and Prof. Dr. Luiz Nacamura Jr for their contributions, corrections an knowledge sharing that taught me a lot and helped me improving this dissertation.

Lastly I want to thank my friends, family, and my girlfriend that inspired and supported me to study once again, and were always there for me during this tough jorney.

Resumo

Dispositivos móveis, como os *tablets* e *smartphones*, ficam mais populares a cada dia. Esta nova classe de dispositivos está evoluindo constantemente em suas capacidades computacionais, permitindo os usuários trocarem computadores maiores por estes dispositivos móveis menores. Diariamente novas aplicações para estes dispostivos são desenvolvidas e publicadas em repositórios de software, possibilitando que os ususários instalem e se beneficiem de novos aplicativos rapidamente. Para gerenciar múltiplas requisições de recursos de diferentes aplicações rodando em paralelo, o design de gestão de recursos do sistema operacional desempenha um papel chave. A vasta maioria dos gerentes de recursos disponíveis nos dispositivos móveis são projetados para maximizar o desempenho do sistema como um todo, nem sempre levando em consideração as características dos processos solicitando recursos na camada de aplicação. Este tipo de estratégia pode levar a casos onde algumas aplicações experimentem tempos de latência mais longos que os desejados durante os accessos aos recursos, comprometendo a Qualidade de Serviço ou a segurança do sistema. Este trabalho tem por objetivo oferecer um novo modelo de gerência de recursos para dispositivos móveis, permitindo a criação de regras que priorizem a alocação de recursos para aplicações específicas de acordo com suas necessidades. Além disso foi implemetado um prototótipo como prova de conceito deste modelo, usando a plataforma Android, para confirmando a viabilidade do mesmo.

Palavras-chave: Dispositivos Móveis, Gerenciamento de Recursos, Desempenho.

Abstract

Mobile devices, such as smartphones and tablets, are becoming more popular everyday. This new devices' class is constantly evolving on what concerns computing capabilities, allowing users to switch from larger computers to smaller mobile devices in several cases. Everyday new apps for these devices are developed and published in software repositories, enabling users to quickly install and benefit from new software. In order to deal with the multiple resource requests from different applications running in parallel, the operating system's resource management design plays a key role. Most resource managers available to mobile phones are designed to maximize the performance of the system as a whole, not always taking into account the characteristics of the processes requiring resources in the application layer. This type of strategy can lead to cases where some applications experience longer latencies than desired while accessing resources, compromising the Quality of Service or system security. This work aims to offer a new resource management model for mobile devices, allowing the creation of rules that can prioritize the resource allocation for specific applications according to their needs. Also it implemented a proof of concept prototype version of this model, using the Android platform, confirming its viability.

Keywords: Mobile Devices, Resource Management, Performance.

Contents

List of Figures

List of Tables

List of Symbols

U	Upper Limit
L	Lower Limit
K	Group of applications K
J	Group of applications J
J	Resource R
L_J	Lower Limit associated with group of applications J
U_K	Upper Limit associated with group of applications K
π	Pi

Acronyms

SMS	Short Message Service
GPS	Global Positioning System
IT	Information Technology
ICC	Inter-Component Communication
UID	User Identifieer
cgroups	Linux Control Groups
LMK	Low Memory Killer
OOMK	Out-of-Memory Killer
VM	Virtual Machine
SIM	Subscriber Identity Module
URI	Uniform Resource Identifiers)
BYOD	Bring Your Own Device
ADB	Android Debug Bridge

Chapter 1

Introduction

1.1 Contextualization and challenges

Mobile devices are becoming more used everyday, receiving growing attention from both the market and academia. New applications are developed daily targeting smartphones and tablets, enabling users to perform more tasks with them as time goes by. These devices provide competitive advantages to companies with respect to data availability [Thomson, 2012] Mobile devices are now a viable work option, being preferred over larger laptops and desktops in several cases. As a consequence, the number of applications running in parallel and competing for resources on these mobile devices is constantly increasing.

As there are more and more mobile devices' users and developers, the number of applications and the volume of sensitive data (e.g. e-mail, contact lists, internet banking, SMS, GPS location etc.) is increasing greatly on the mobile devices community. At the same time, the concern about mobile devices' resource management, system responsiveness and security features becomes bigger. The possibility of data leakage is becoming a threat to the users, and it has been already studied by the academia in several works such as [Schreckling et al., 2012, Vidas et al., 2011, Felt et al., 2011c]. Also, as the bandwidth and processor capacity of the mobile devices continue to improve, mobile devices are becoming more interesting targets to malicious attackers that could benefit from these capabilities (e.g. botnets, data leakage, etc.) [Oberheide and Jahanian, 2010].

Currently operating systems provide little to implement usage control rules [Russello et al., Oct] or enforce different policies for different users or applications [AirWatch, 2012]. The software market already offer some tools to mitigate security risks like MDM (Mobile Device Management) tools, anti-viruses and improve system performance. However currents Operating Systems do not assure that these security applications will always respond on a timely fashion, or that they won't suffer from resource constraints. In the Android Operating System for instance, when the system is running short on resources, applications can be terminated through *Low Memory Killer* (LMK) and *Out-of-Memory Killer* (OOMK) mechanisms [Lim et al., 2013]. By default, this platform does not provide any sort of customization tools to the users, on what concerns how resource requests should be treated among different applications.

In order to deal with a vast number of applications, the OS designers opted by an aggressive resource management strategy to increase system performance and throughput in general. This approach does not evaluate characteristics of the resources requests, and trusts blindly the priority defined by the developer to their processes. In a scenario with too many applications it is possible that this strategy would allow some low importance applications to consume a significant part of the resources available, impairing the execution of applications of higher importance. By having critical applications response times affected, the overall security of the mobile platform could be compromised or the user could experience undesired latency times when accessing his applications.

The existing resource managers are typically designed to increase the system throughput by handling requests in a fair manner, among all applications. That means that the application criticality might not be taken into account on resource requests. As consequence any application could potentially compromise the resource availability to critical applications. Malicious applications could exploit this weakness by over requesting resources, increasing the applications response time, impacting both the system security and Quality of Service (QoS) [Kim et al., 2011].

This work is intended to propose alternatives to this resource management model. While not ensuring maximum system throughput, the idea is to ensure that applications that aren't critical to user needs have a limited resource consumption, and that applications perceived as more important will always have a minimum of resources allocated to them, in order to meet the desired performance expected by the user, reduce latency times to important applications, adequate scheduling to time sensitive applications and increase platform security overall.

1.2 Main Goal

This work aims to present a new resource management model to mobile devices. This model will allow the system to prioritize the resource allocation to specific applications, according to their requirements. Although several resources (processor, memory, network bandwidth, disk space, and disk input/output operations) can have a significant impact on the application run times, this work will focus on the processor resource. Based on the definitions set in this model, applications will have delimited chances of getting a specific processing resource allocation, that can be increased or decreased according to the usage rules specified by the user.

This model will also provide means to set up maximum or minimum usage threshold levels, limiting the usage for specific applications to specific resources. Through this model, the theoretical basis for new resource managers will be available, providing a scheme that will allow schedulers to customize which applications will have a greater priority when accessing resources, or which applications shall not surpass a predetermined percentage usage of the available resources.

1.3 Specific goals

- Define an resource management model that allows mobile devices to implement new resource management rules in order to privilege or limit the access of determined processing resources to specific applications. These rules can be adjusted dynamically during usage and will last during all application process' execution.
- Support the creation of new model that allow customizations of the scheduling the processor resource. This model should be implemented through the definitions of rules that can be dynamically adjusted.
- Implement a proof of concept prototype capable of controlling processing resources dynamically, through upper/lower limits rules that can created and adjusted on the fly.
- Evaluate overhead and the viability of the implemented prototype.

1.4 Contribution

This work creates a model that enables mobile devices to handle resource management differently, providing a theoretical model that can be used to better adjust the resource management based on application requirements. For instance this model can be used to ensure that a certain piece of software will always have enough resource throughput to execute properly. This can be particularly helpful to ensure that critical applications are always running properly, or that certain applications will provide a timely response to the user or system. . While this model does not provide any means to determine what are the appropriate resource requirements for an application, it does provide means to implement new resource management rules, that can ensure proper processing resource allocation, once these requirements have been defined. How these application requirements would be defined is out of the scope of this work.

1.5 Document Structure

This document is divided in seven chapters. In chapter 1 Introduction, is offered a presentation of some resource management issues of the current mobile devices platforms, the goals of this work and its contributions. Chapter 2 deepens our understanding on the current resource management strategy for mobile devices, and discuss some related work to improve resource management issues. Chapter 3 discusses in more detail the Android platform, that is being used for the implementation of the prototype of this work. Chapter 4 presents a new resource management model, that provide means to better control resource allocation by application, taking into consideration application requirements. Chapter 5 details a proof of concept implementation, intended to prove this new model viability, discusses this work methodology, and results. Chapter 6 summarizes this work accomplishments, future work and conclusions.

Chapter 2

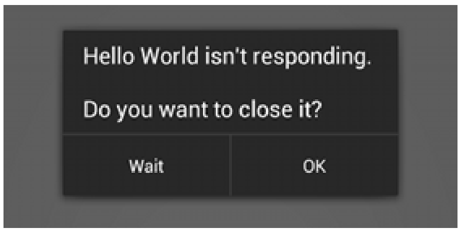
Mobile Devices Resource Management Model

Due to the nature of mobile devices, their resources and resource management differ a little bit from traditional computing environments. Most of the popular traditional computing environments, such as desktops are implemented through the CISC architecture and typically can rely on a standing power supply. Now most of mobile devices are implemented by the RISC architecture that be implemented directly on silicon. The intention is to make smaller computing systems that provide mobility to the users. Mobile devices typically rely on SD cards and are designed to not consume much power, as instead of having a power supply most of it is usage is battery constrained.

This chapter discusses characteristics of most of the current resource managers, that are used in the mobile devices platforms. It also reviews some of the existing state of the art work, that is being applied to the existing mobile devices platforms, in order to improve some resource management features.

2.1 Current Resource Management Model

Using a mobile device that does not respond to the user touches on timely fashion can be perceived as very frustrating to the user. Due to issues like that, mobile devices operating systems tend to prioritize applications running on the foreground (visible to the user) rather than the ones running on the background. Applications suffering from resources starvation might even be terminated in some cases, to provide better user experience. In Android mobile devices, for instance, the operating system reacts against applications that are unresponsive for a period of time, by popping up a dialog box stating the app has stopped responding, such as the dialog in Figure 2.1. If the application has not been responsible for a considerable period of time so the system offers the user an option to quit the app or even kills the application.


Hello World isn't responding.

Do you want to close it?

Figure 2.1: Application Not Responsive pop-up in Android platform. Source: Android Developers

Although this sort of strategy can help to improve the user experience when accessing applications in the foreground, it can also impair applications running on the background, due to lack of resources, in some cases even closing applications. Applications running low on resources on the background can be bogus or not have appropriate response times when interacting with another system components. Such applications could be critical applications, responsible for the system security or important information.

Typically the mobile devices' resource management policies are designed in order to increase the system throughput, by handling requests in a fair manner (provided they are running in the same context e.g background), among all applications. That means that the application characteristics are not taken into account on resource requests. As a consequence, any application could potentially compromise the resource availability to critical applications. By not having an specific limits on the resource usage, one or more applications can exhaust some system resources, force some applications to be killed or to present an undesirable response times. This can affect directly some security mechanisms in place or the overall QoS[Kim et al., 2011].

Although the current access control models are specific at some extent on which resources are being accessed, they are very limited when controlling the usage of such resources. For instance in the Android platform, the resource access is based on the Linux Kernel schematic, permissions are explicitly granted to the resources (read, write and execute) only once at install time. It does not allow permissioning customization, implementing a all-or-nothing approach while granting resource accesses for applications, during install-time [Felt et al., 2011a].

In iOS, typically there is no usage of explicit permissions. All applications that are published in Apple App store must go through a review process [Apple, 2013]. If an application passes the Apple App store review process, it is assumed that it is a safe app. Once an application is installed, there is no way of restricting most of the actions that an application can perform. Unlikely Android applications, iOS applications do not reveal which permissions or which resources the applications will use. All the applications have the same access to the iOS mobile device resources and can use them without the user's permission. That being said, once an application has been installed on the device, it can perform any operation without taking into account the characteristics of other operations running simultaneously.

To sum up, in both major mobile devices platforms, users or administrators are not able to customize rules for prioritizing resource accesses to certain applications. Due to the lack of resource management control, there are cases where some applications experience longer latencies than desired while accessing resources, compromising the QoS or the system security.

For example, the Android device manager uses the *Completely Fair Queuing* (CFQ) elevator algorithm for disk I/O management. CFQ is intended to ensure fair allocation of the disk bandwidth among all the applications performing disk I/O requests, and to increase the system throughput [Kim et al., 2011]. This model does not take into consideration the application criticality when scheduling disk I/O activities, what could lead to some undesired wait time for disk I/O in some applications. Also when system is running low on memory, some low priority applications might be terminated by *Low Memory Killer* (LMK) and *Out-of-Memory Killer* (OOMK) operations [Lim et al., 2013]. The current platform does not allow the user to choose which applications will get a higher memory amount. This can be particularly critical as due the nature of SD cards most mobile platforms do not support disk swaping.

Another resource that can be utilized to illustrate this is the network subsystem. In Android, the network subsystem is implemented using two queues. An Input queue for incoming packets and an output queue for outgoing packets. The default Android implementation for these queues is the *First Come First Served* (FCFS) strategy. In some implementations it can incorporate policies that improve QoS determining which is the proper order that packets are served [Kim et al., 2011]. Although this design should maximize system throughput that can be inconsistent with some application requirements, impairing applications that might require a broader bandwidth while competing with other applications.

Another example is the disk usage, where there are no limits pre-defined. Mobile devices typically do not implement quota rules to limit disk usage, potentially enabling applications to consume all space available that could be required for more critical operations.

2.2 Related Work

There are several attempts to improve resource management mechanisms for mobile devices. In [Kim et al., 2011] were performed modifications the block device scheduling layer and network packet handling layer of a Linux kernel version 2.6.35 (Android 2.2). Named Aciom, this modification allowed differentiated disk and network I/O management, for some applications avoiding delayed responses for time sensitive applications.

In this work, the strategy was to classify applications as time-sensitive, bursty or plain according to the system services they use (e.g. media player), or whether applications are running in foreground or background. For time-sensitive applications a certain amount of disk and network I/O is guaranteed to the OS. For a bursty application the maximum I/O bandwidth is allocated to the application. For plain applications no resource allocation is guaranteed. Although this approach identifies some time-sensitive applications it does not encompass all applications, and it does allow prioritization of applications explicitly defined by an user or administrator.

In [Lim et al., 2013] a new memory partition schematic was developed in order to avoid LMK and OOMK operations in Android mobile devices. Their strategy consists on dividing the memory in two different areas, called virtual nodes. Virtual node VNODE 0 reserves a bigger chunk of memory resources for native or Google trusted applications while VNODE 1 is destined for untrusted applications (black market). By doing that they prevent non-trusted applications from exhausting the entire memory, reserved for allegedly critical applications. With that this paper claims to have huge improvements on avoiding LMK and OOMK operations (none happened in their experiment) and 20 times as much as free memory with this schematic. This work was performed on top of Android Ice Cream Sandwich 4.0.4 and Busybox 1.18 to Samsung SENS R60+ (CPU: Intel Core2Duo, MEM: DDR2 2G) laptop but it does not details which applications were running at the experiment results.

In [Starke, 2006] a model for controlling the amount of resources available for applications was presented. Although this worked focused on a desktop architecture instead of mobile devices, it presents a model capable of defining and adjusting limits of resources utilization for an specific groups of processes associated to either an user, groups of users or processes associated with an specific application. Similar concepts are adopted in this work. It also does allow memory prioritization for applications in order to meet application resource requirements.

In [Murmuria, 2010] *Linux Control Groups* are used to develop a mechanism that provides a sort of measurement of energy consumption by individual process in the Android Platform. By doing that they allow the user to understand what process are most battery consuming and how consuming they are. Although a light weight framework as created for measuring this battery usage, no control in the energy consumption of these processes is offered yet.

In [Colmenares et al., 2013] an adaptive resource management system is developed. This work implements a discovery mechanism that identifies the correct mix of resource assignments necessary to achieve QoS application requirements. Resources are distributed to QoS domains called cells, containers that ensure resource distribution in parallel, with user scheduling of resources inside the cells. The allocation and distribution of resources to cells is constantly adjusted in order to meet QoS requirements efficiently. However defining these requirements is not be trivial. Although this work is not focused on mobile devices, similar concepts could be applied when working with resource management mobile devices.

In [Muhammad Shiraz and Ahmed, 2012] it presented SmartSim. It is toolkit for modeling the application processing capabilities in Mobile Devices. SmartSim simulates thes operating system behavior of mobile devices. It provides a mechanism to determine resource utilization on mobile devices during the execution of applications.

In [Chatterjee, 2013] tools are developed to better understand memory scheduling and proposes innovations for CPUs and GPUs. This work proposes novel memory scheduling techniques that are that take into consideration the client access patterns and the microarchitecture of the memory device. Also new processing scheduling techniques are proposed tom improve graphics processing.

2.3 Discussions

The Mobile Device's resource management strategy is designed to increase the system throughput as whole, not taking into consideration the applications' requirements. This resource management model can be susceptible to problems, potentially making applications suffer from resource starvation, sometimes compromising the system correct functioning, or security. Examples include, security products, that might need scan or hash files on the fly to ensure applications are legit ot thar have not been tampered. If the security feature does not have enough processor capability to verify the file on a timely fashion, the user might experience longer latency times or the application might even be terminated, regardless of being legit or not. Possibly anti-virus could not detect threaths on timely fashion. Also applications that are time sensitive might be impaired if competing with applications that had priority poorly set. For instance high quality media reproduction requires a considerable amount of the processing resources to be properly reproduced. Other trend is developing on mobile devices. Prioritizing processing allocation for compilers might allow the user a better overall experience, when the code is being compiled on the background (e.g. processes running in background get less processor time by default in the Android plaform). Besides that developers might inadequate priorities to their processes and threads, impairing the response time for certain applications that might be critical to the utilization context. There are several attempts in the academia to improve the way that resource managers distribute resources to the applications. In chapter 4 it will be discussed the model proposed in this work. But first it will be reviewed the Android platform in 3

Chapter 3

Understanding the Android Platform

This chapter will discuss some features of the Android platform, the base operating system of choice to prototype the model defined in the chapter 4.

3.1 Android Overview

Android is an operating system designed mainly to be used by mobile devices such as smartphones and tablet computers. It was developed by a group of companies known as OHA (Open Handset Alliance), a consortium which is led by Google. Other OHA members that can be highlighted are HTC, Sony, Intel, Motorola, Qualcomm, Dell, Texas Instruments, Samsung Electronics, LG Electronics, T-Mobile and others [OHA, 2013]. The OHA's purpose when developing Android was to create a shared product that would allow each contributor to be able to adjust the resulting product to its own needs, while preventing other contributors from restricting or controlling the innovations of each other.

In order to accomplish that, Android was created as an open-source platform encouraging more companies and developers from the smart-phone community to contribute. This development strategy made Android the most popular mobile platform in the world, with more than one million devices being activated daily [ANDROID DEVELOPERS, 2013]. The number of applications available in Android already surpasses the number of application available for iPhone on the Apple App Store [Distimo, 2012] being estimated as over six hundred thousand applications available in Google play.

Being the most popular platform for mobile devices, Android is receiving growing attention from the mobile community, as new applications are being released constantly for their use. As a consequence, there is a increase on the amount sensitive information stored on these devices. It can be also noted that the hardware and software evolution are making mobile devices more powerful, both bandwidth and processor wise. As a direct consequence these devices are being transformed in valuable targets for attackers, that could possibly benefit from these them if they can gain some sort of access [Oberheide and Jahanian, 2010]. In the following sections it will be discussed some of the features that the Android platform has in order to protect the mobile devices.

3.2 The Android Archtecture

The Android platform can be perceived as a software stack that includes an operating system, middleware, applications. The operating system is based on a Linux Kernel, which is responsible for the hardware abstractions such as drivers, memory management, process management, network and several other security controls (such as user management and file access). The middleware layer is responsible to provide services required from the application layer. It contains the Android native libraries, that implement several different functionalities such as database access, integrated browser, media support, graphic interface access, and others. On top of that there is the application layer made of compiled java code. This layer can be divided on applications framework and application [Shabtai et al., 2010].

In the application framework are found some of Android tools and services such as content providers, activity manager, telephony manager, location manager and others. On top of that there is the application layer that is where are the user's application like phone, e-mail client, browser, games and others. All applications are isolated on their own space, running processes with their own, unique, and low privileged user id, on top of their own instance of a Dalvik Virtual machine [Felt et al., 2011a]. If one application is bogus or crashes, the failure should not propagate to other applications. There are specific permissions for each resource that is required for an application. Permission requests that were not explicitly granted to an application are denied by default. By default, applications can not access directly other application or OS spaces.

Every Android application runs isolated on their own user space on top of a virtual machine. The Dalvik technology allows the mobile devices to run several virtual machines at the same time efficiently. The Dalvik VM relies on the kernel for underlying functionality such as threading and low-level memory management. The application framework is made of compiled code for the Dalvik Virtual Machines and is responsible for different services and components that are used by the different application that ran on top of them. On the upper level are the key application that are pre-installed in advance by the manufactures, and the ones that might be added by the user.

3.2.1 Android Security

The Android relies on three key strategies to enforce the Security on the Devices:

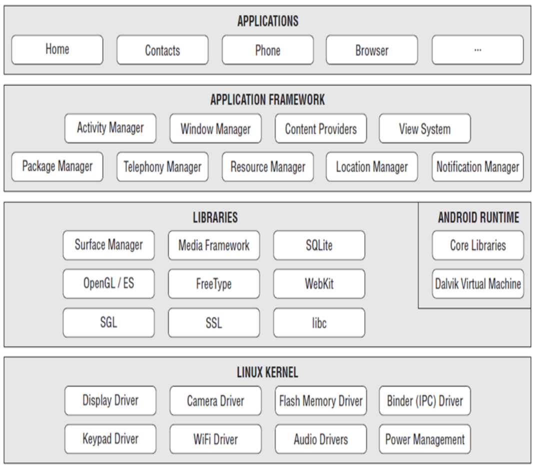


Figure 3.1: Android Diagram Block Architecture. Source:[ANDROID DEVELOPERS, 2013]

- Application Isolation: Every application space is isolated through a Dalvik Virtual Machine. That means that a application can not access directly other application or OS spaces. If one application is bogus or crashes, the failure shall not propagate to other application.

- Fine Grained Resource access: There are specific permissions for each resource that is required for an application. Permissions request that were not explicitly granted to an application are denied by default.

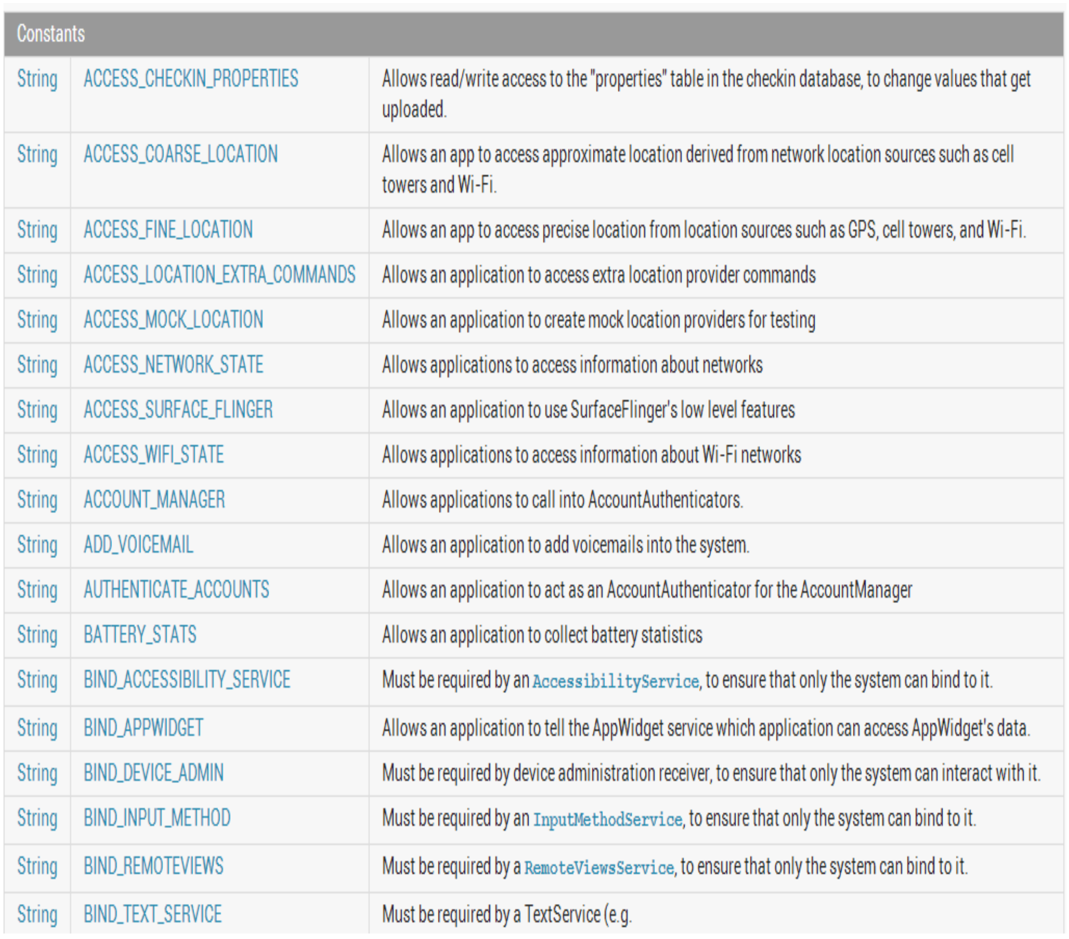
- Linux Kernel: Android is built on top of a Linux Kernel. Every application is treated as a different user (UID) and for some resources (e.g. Internet, Bluetooth) the Linux permission/groups schematic is applied.

Android provides a fine grained permission manifest that allow both the user and the developers to understand which resources one specific application has access to. Up to current date there are 130 default permissions defined by the Android Permission Manifest in order control the access to the mobile devices' resources.

Android also allows the developers to define their own set of permissions to their own application resources.

3.2.2 Android Component Types

The Android platform forces the develops to use a different approach of structuring their applications, [Enck et al., Feb] instead of having a starting main function, the developers must design their application using components. There are four basic components in the Android applications that will be detailed on the section below:



Constants		
String	ACCESS_CHECKIN_PROPERTIES	Allows read/write access to the "properties" table in the checkin database, to change values that get uploaded.
String	ACCESS_COARSE_LOCATION	Allows an app to access approximate location derived from network location sources such as cell towers and Wi-Fi.
String	ACCESS_FINE_LOCATION	Allows an app to access precise location from location sources such as GPS, cell towers, and Wi-Fi.
String	ACCESS_LOCATION_EXTRA_COMMANDS	Allows an application to access extra location provider commands
String	ACCESS_MOCK_LOCATION	Allows an application to create mock location providers for testing
String	ACCESS_NETWORK_STATE	Allows applications to access information about networks
String	ACCESS_SURFACE_FLINGER	Allows an application to use SurfaceFlinger's low level features
String	ACCESS_WIFI_STATE	Allows applications to access information about Wi-Fi networks
String	ACCOUNT_MANAGER	Allows applications to call into AccountAuthenticators.
String	ADD_VOICEMAIL	Allows an application to add voicemails into the system.
String	AUTHENTICATE_ACCOUNTS	Allows an application to act as an AccountAuthenticator for the AccountManager
String	BATTERY_STATS	Allows an application to collect battery statistics
String	BIND_ACCESSIBILITY_SERVICE	Must be required by an `AccessibilityService`, to ensure that only the system can bind to it.
String	BIND_APPWIDGET	Allows an application to tell the AppWidget service which application can access AppWidget's data.
String	BIND_DEVICE_ADMIN	Must be required by device administration receiver, to ensure that only the system can interact with it.
String	BIND_INPUT_METHOD	Must be required by an `InputMethodService`, to ensure that only the system can bind to it.
String	BIND_REMOTEVIEWS	Must be required by a `RemoteViewsService`, to ensure that only the system can bind to it.
String	BIND_TEXT_SERVICE	Must be required by a TextService (e.g.

Figure 3.2: Snip from Android Permission Manifest Documentation. Source: [ANDROID DEVELOPERS, 2013]

Activity

Activity components can be considered as the presentation layer of the applications. Typically developers create one activity per screen that is needed by the application. At any given time only one activity on the system can have keyboard and processing focus, keeping all other activities suspended. Activities can exist in three stages:

- Resumed or Running: The activity is in the foreground of the screen and has user focus.
- Paused: Another activity has focus, but this one is still visible (partially covered). A paused activity is completely alive and its object is kept on the memory.
- The activity is completely obscured by another activity (background).

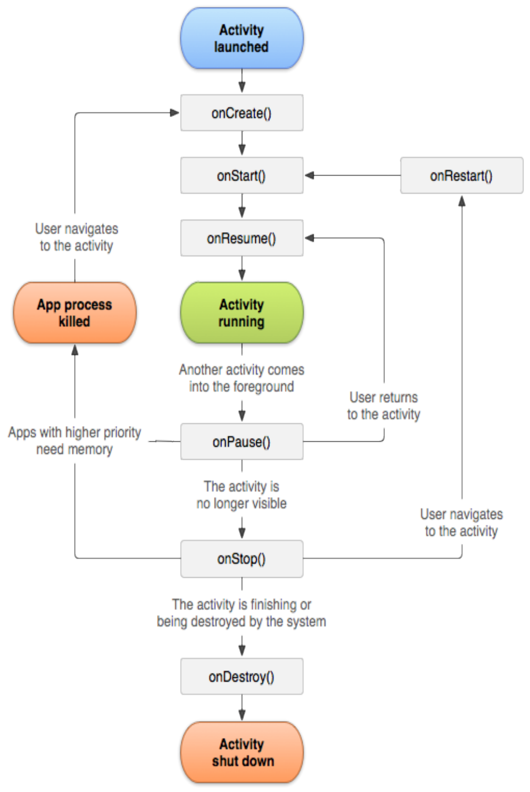

Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 3.3: Activity Lifecycle Workflow. Source: [ANDROID DEVELOPERS, 2013]

Service

Services are application components that perform processing on background mode. Very often activities can start services when the action must persist after the user focus is switched to other activity (e.g. downloading a file, playing a music, checking updates, etc.). Services can be either started or bound. If a service is started, it runs independently from other components, even if the component that started it is destroyed. Typically, a started service executes only a single operation and does not return a result to the calling service. If a service is bound to another component, it runs only as long as this application component runs. Through inter-process communications, bound services can interact with the component that started them, getting or sending information through a client-server interface.

Content Provider

When applications need to make data available to other applications, they must set up content providers. Content providers are used to both share and store data through a relational database interface. The access to their content is provided through URI (Uniform Resource Identifiers) that can grant both read or write access to their database. The access from other

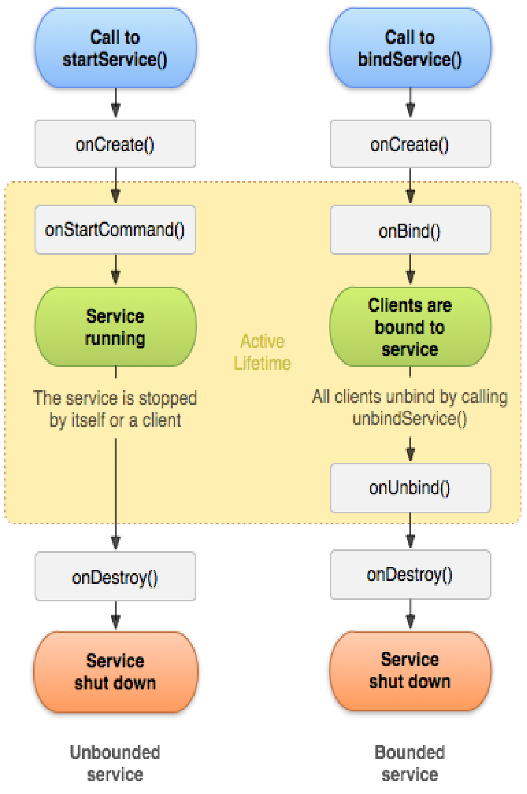


Figure 3.4: Service Lifecycle Workflow. Source: [ANDROID DEVELOPERS, 2013]

components to content providers can be made through SQL queries if authorized. Content providers can be set up in two different ways:

- File data: Stores files in application private space, and offers handles to the files.
- Structured data: Uses databases to store date through SQL-like queries.

Broadcast Receiver

In order to receive messages from other applications, the developers must create broadcast receivers for their own application. They work similarly as mailboxes, that receive messages that were broadcast from other applications. Broadcast receivers can subscribe to destinations where broadcast messages are sent. Also, applications can specify a broadcast receiver by including the *namespace* assigned to it. Broadcast Receivers are registered dynamically or statically in the application manifest. They are able to subscribe to destination of interest where messages are sent. There are two major broadcast classes:

- Normal broadcasts: Completely asynchronous but more efficient.
- Ordered broadcasts: Delivered only to one receiver at a time. Controlled by the priority defined by the receivers.

3.2.3 Android Permission Protection Levels

The Android platform provides fine-grained permission control on what concerns which resources are available for a specific application. Developers need to explicitly request access to the resources that the application will use. These permissions are classified in threat levels. According to [Felt et al., 2011b] there are four threat levels:

- *Normal*: Permissions from this category should not lead to harmful consequences to the user but they still can be annoying. Permissions from this category can be viewed during the application install but they are hidden behind a collapsed drop-down menu by default. Examples include setting the alarm, setting the wallpaper, accessing the information about available WiFi networks and vibrating the Phone.

- *Dangerous*: Access to API calls that could lead to harmful consequences to the user. Permission requests from this category are explicitly shown to the user at the time of application install. Examples include Internet Access, Recording Audio, and Using the Camera.

- *Signature*: Permission category that automatically grants access to application that have been signed with the device's manufacture certificate. All requests from applications that do not contain this signature will be ignored. Typically the most sensitive permissions are protected with this kind of permission. Examples include: the ability to inject user events and test hardware.

- *SignatureOrSystem*: This category grants permissions for applications that are installed on the system image folder (*/system/app*) or signed with the same certificate as the application that declared the permission. Examples include the ability to backup, setting the time or deleting packages.

3.2.4 Android Security Enforcement Mechanisms

Security enforcement in the Android platform is implemented at two different levels: each application executes as an unique user identity (UID), that receives system-level isolation by underlying Linux system; and the Android middleware contains a reference monitor, called *ActiveManagerService*, that is responsible for the ICC: *Inter-Component Communication* [Felt et al., 2011a].

3.2.5 ICC: Inter-Component Communication

All application communication on the Android platform is done through some special kind of message called *intent*. Intents are sent through a mediator called ActiveManagerService.

These intents are sent either if the application has the proper permissions for sending the desired message (defined on the permission manifest) or if the UID corresponds to a system process. If none of the conditions previously mentioned are met, intents can not be sent from the application. Applications might also require permission to receive intents so messages broadcast to other applications are not seen by unauthorized ones. Some malicious applications might attempt to delegate their intents to other application by writing information on the SD drive for instance, so there are some work ongoing on the direction of monitoring and enforcing policies on any possible communication between all the applications [Schreckling et al., 2012]. Other proposed solution to this issue is to limit the kind of intents each application can send would be having the user to decide which resources will be granted to the application at installation time [Nauman and Khan, 2011].

3.3 Authentication

The Android Security Model benefited from authentication schematics that were inherited from older platforms [Shabtai et al., 2010]. Typically, mobile devices have functionalities and attributes in order to identify users, monitor and control usage (calls, Internet, SMS, etc.) so the service carriers can charge the users accordingly. Being a mobile device platform, Android takes advantage of the security features of the classical cellular phone design. The authentication is performed via SIM card and related protocols in most cases.

Besides the SIM card authentication Android provides other security features to authenticate the user so the device can be used. Following below an analysis of the four most commonly available screen lock options [Braga et al., 2012]:

- facial recognition: Although there are researches to improve the security of this method, it is considered the most insecure one, as it can be easily bypassed by malicious users using pictures of the user that we are willing to authenticate.

- Pattern lock: This method requires the user to draw a pattern in nine dots (3x3 field). These dots can be touched and dragged one dot at a time (redundancy input is not available). This approach is considered fairly vulnerable as it is possible to look for a trail left in the user's screen. The problem is that the amount of drawing combinations is limited as you can not repeat one dot.

- PIN numbers: Blocking through a numeric password. The security on this approach is limited due to the fact that only numbers are used. This method could possibly be broken on a brute-force attack.

- passwords: It is still considered the most secure blocking method as it allows the user to combine letters, number and symbols on its own password, creating a bigger range of password possibilities.

Too many failures on unlock attempts can result on a preventive lock that will require the user to authenticate through the Google Account so the phone can be unlocked once again.

Application authentication is pretty much based on checking if the application signature matches the developer certificate. This prevents non legitimate packages that might have been tampered to be installed. Also developer have authentication protocols to validate their application in running state (e.g. OAuth 2.0 for Android devices running 2.2 or later). In the Android platform all applications are treated as one user that have one unique user identifier (UID).

3.4 Android Kernel x Linux Kernel

The Android Kernel was based in a regular Linux Kernel though several modifications were performed to adequate its Kernel to the mobile devices usage. These changes implement both large and small additions, from the addition of a different flash file system (YAFFS2), to very small patches to augment Linux security (paranoid networking patches). Other changes aimed to provide a better power management as typically Android mobile devices are battery constrained. Besides that some changes were intended to improve performance, for instance the Android-specific inter-process communication mechanism, and the remote method invocation system that are known as binder.

3.5 Android Resource Management

The Android resource manager is very similar to the one implemented on the regular Linux kernel. The goal is to increase the system throughput by handling requests in a fair manner, among all applications. For instance the disk I/O scheduling implements CFQ, the network subsystem is implemented through an Input queue for incoming packets, and an output queue for outgoing packets. These queues follow First Come First Served (FCFS) strategy. Now the processor scheduling is based in mainly in two definitions of the process: the process priority and which cgroup it resides.

3.5.1 Processes'priority

The amount of processed time assigned to task might be greatly affected by its priority in some cases. All process are launched with a priority (niceness) that is defined by a range of

-20 (the highest) to 19 (the lowest). The higher the value, the lower is the priority. Developers can explicitly set a process priority in its code based on the application requirements. It is possible to have the priority of a process or thread during execution time though that is unusual. If the priority is not set in the application it starts with the default value = (0).

The percentage of scheduling resource should be approximately proportional to *20 - p*, where *p* is the process priority. For instance a process with priority 0 should receive about two times the amount percentage of the process with priority 10, provided that they are part of the same cgroup. Android predefined some constants that facilitate determining the priority adequated to certain apps such as:

- THREAD_PRIORITY_URGENT_AUDIO (-19)
- THREAD_PRIORITY_LESS_FAVORABLE (1)
- THREAD_PRIORITY_DEFAULT (0)
- THREAD_PRIORITY_AUDIO (-16)

However the cgroup to with the process is associated might minimize the effect of such priorities. A running in the background cgroup for instance might get less CPU time than another tasks with lower priority running in the foreground cgroup.

3.5.2 Linux Kernel Control Groups

Android takes advantage of Linux control groups to perform resource management activities. Linux control groups are part of a Linux kernel feature that provides means for aggregating or partitioning a collection of applications and their future child processes and threads into hierarchical groups of specialized behavior [Jackson and Lameter, 2013]. In other words, control groups implement sets of process that are bound by the same criteria. Control groups can be used to enable resource limiting, prioritization, resource usage accounting and control. They have a considerable low overhead when compared to other techniques (e.g. virtualization, jailing) [Murmuria, 2010].

Control groups are managed through user-space operations, and can control access to CPU, memory, network, input/output devices, and the file system. This management allows to place in frozen or suspended state process that are resource exhausting or that appear to be malicious. As this management system was originally created to support traditional computing platforms, its original version does not provide battery support, though is there some research in place that extends its capabilities to provide some battery support [Murmuria, 2010].

In Android, Cgroups are used to prioritize processing resources for applications running in foreground. There are two cgroups that control resource allocations, one for the applications running in foreground an other for the applications/services in background. By defaults, the foreground cgroup reserves 95% of all processing resources to foreground apps. The Activity Manager services is responsible dynamically to update a list of tasks IDs running in foreground, so as soon as the applications get moved to the user screen they get privileged access to resources. The other 5% of the processing resources are destined to applications or services running on background.

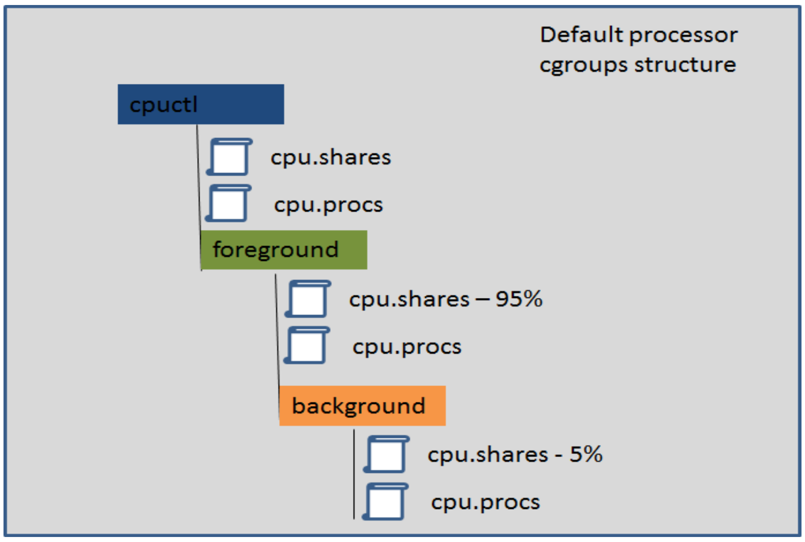


Figure 3.5: Representation of default Android Cgroups structure

Typically Android kernel is configured to utilize control groups scheduling, that can be accessed through the file system as a folder structure, mounted under /dev/cpuctl. Folder apps represent the foreground apps control group. Inside this folder there is bg_non_interactive that corresponds to the background control group. Figure 3.5 presents a representation on how the standard Android cgroups structure is designed. There are few files to set up control groups but there are two that are the most concerning for this work, File cpu.shares that controls the amount of resource being assigned to the control group and file cpu.process control the processes added to the control group. It is also possible to assign specific cores are assigned to a cgroup through cpuset configuration.

3.6 Discussion

This chapter provided a review of the Android platform discussing their security features, access control, applications life cycle and processing resources prioritization. It shows that by attempting to maximize the overall resources throughput to foreground applications, the Android operating system might not attend some applications requirements.

In order to address these issues a new resource management model is presented in chapter 4, that is intended to provide means to adjust the processor scheduler to different user requirements. As Android is the most used mobile device OS and it is open source, this work decided to adopt android as a host for the validation of the viability of this new resource management model, processe detailed in 5 .

This model should provide the ability to ensure a greater throughput to applications with respect to which resources are required by them. Through the concepts of upper and lower limits rules, that represent the maximum and minimum resource usage that an application will have assigned to itself. Through this modification, resource management policies can set up to overcome some of the current usage control and resource management limitations that were discussed, present in most current mobile platforms.

Chapter 4

Conditional Resource Management Model

4.1 Proposal: Conditional Resource Management Model

In order to ensure that a single application or a group of applications do not surpass a certain processing usage threshold, that could compromise the resource availability to other applications, or impair the system stability, this work proposes the creation of usage limits to specific resources. It supports the implementation of conditional resource management rules, through the creation of two resource usage limits, *the upper and the lower limits*.

The *upper limit* represents the maximum utilization allowed, by a single application or a group of applications, to access a specific resource (set of processing units). Whenever the threshold of maximum usage is achieved, the resource manager adjusts the resource allocation to not exceed the defined limit. This limit is intended to prevent excessive resource usage from specific applications.

The upper limit U, controlling resource R, for a group of applications K, is a percentage value between 0% and 100% that is represented by U_K. If $U_K.R = 0\%.R$ that means that the resource is denied for K. The formula below show the definition limits for a generic upper limit U_K when controlling resource R (that can be either the whole processor or a subset of cores).

$$[0\%.R \leq U_K.R \leq 100\%.R]$$

The *lower limit* represents the minimum utilization required by a single application or a group of applications to access a specific set of processors. For rules of this type, the resource manager ensures that at least the amount of resources corresponding to the lower limit is reserved for the applications to whom this rule is applied.

The lower limit L, controlling resource R, for a group of applications J is a percentage value between 0% and 100% that is represented by L_J. If $L_J.R = 100\%.R$ that means that the resource *R* is exclusively granted to the to the group of applications *J*, even if it is competing against other apps for this resource, as long as *J* is claiming and using all resources *R* available.

$$[0\%.R \leq L_J.R \leq 100\%.R]$$

All existing *m* lower limits combined can never exceed 100% of the total usage of the same resource.

$$\sum_{n=1}^{m} L_n.R \leq 100\%.R$$

The purpose of these limiting rules is to provide differentiated management to resources like processor, memory network and disk. It is acknowledged that battery is a crucial resource in mobile devices components. However this resource will be kept out of this work scope, due to the complexity of measuring power consumption, and little supported provided by the mobile devices operating systems to manage this resource. It can be note that there are some works that are attempting to provide support this kind of control. For instance, in [Murmuria, 2010] a mechanism that provides a sort of measurement to energy consumption by individual process is developed, through the usage of control groups. Android wakelocks and device specific usages are monitored, in order to attribute to each process their share of consumed energy. Although energy consumption profiling is offered in that work, no control over the resource allocation is implemented.

4.1.1 Processor Limits

In order to manage the processing unit, the resource manager needs to adjust the processing scheduling. The strategy here is to change the proportion of scheduled processing time, for applications that have been associated to upper and lower limit rules. According to the setup of these rules, the resource manager will be increasing or decreasing the percentage of processing allocation, to meet applications' requirements, as long as the applications to these requirements have been defined for are running.

Through the lower limiting rule, applied to the processor, a minimum percentage of processing is reserved to a group of applications (one or more) while this group claims processing. For instance, if we have an application running only a single process *P1*, that has a lower limit defined as 50%, at least half of the processing time will be granted to this application, whenever it requires the processor resource. Using this rule, more than half of processing resources can be granted to the application, if there are sparing processing cycles available.

In figure 4.1 an example of this rule is illustrated. In this case process *P1* is competing against processes *P2* and *P3* for the processor resource. Before time elapsed (*t*) reaches moment 2, *t<2* there are no limiting rule is in place, so all process are competing for the processor resource in the same conditions. At moment *t=2*, a lower limit rule that reserves 50% of processing resources to P1 is activated. When this happens within a certain period of time (*2<t<4*), the resource scheduler is adjusted to meet this new rule, so as soon as practical this process *P1* will be receiving a minimum of 50% of the processing resources. After that(*t>4*, this rule ensures only a minimum resource assignment of 50% but it does limit the maximum usage. Whenever resources are available, *P1* might have more than half of the resources assigned to it, while active.

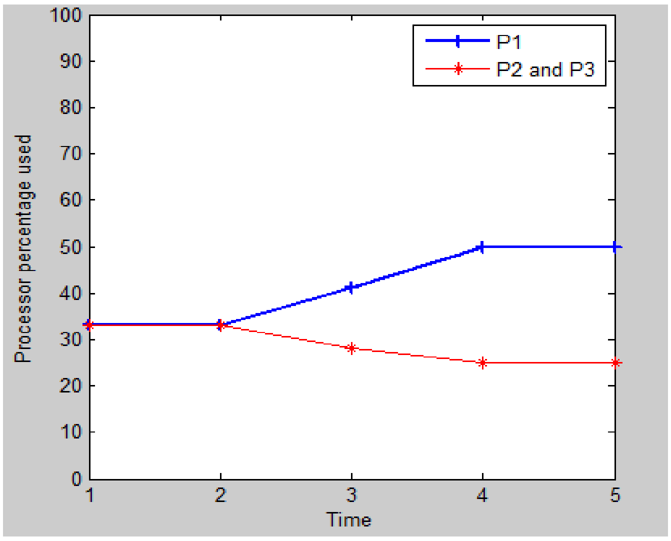


Figure 4.1: 50% Lower limit rule applied to applied to processor usage for process P1, competing against processes P2 and P3

As expected, the active lower limit rules in place, can not exceed 100% of processor reserve combined. If such situation happens the resource manager should re-evaluate the resource requirements from the applications, and adjust the rules'limits, to not exceed 100% of processing reserve.

Using the upper limiting rule for the processor, a maximum percentage of processing is imposed to a group of application during its execution. As a consequence the processor usage for this group of applications will not exceed the threshold defined by the upper limit, by any means. In figure 4.2 an example of this rule is illustrated, where process *P1* is competing in fairness against processes *P2* and *P3* for the processor resource, while *t<2*. At moment *t=2*, an upper limit that limits the maximum usage of processing resources to *P1* in 20% is imposed. When this happens, the resource assignment is adjusted to to not exceed 20% as soon as practical, as shown during a certain period of time (*2<t<4*). After that (*t>4*), process *P1* will not not exceed one fifth of the processing scheduling allocation.

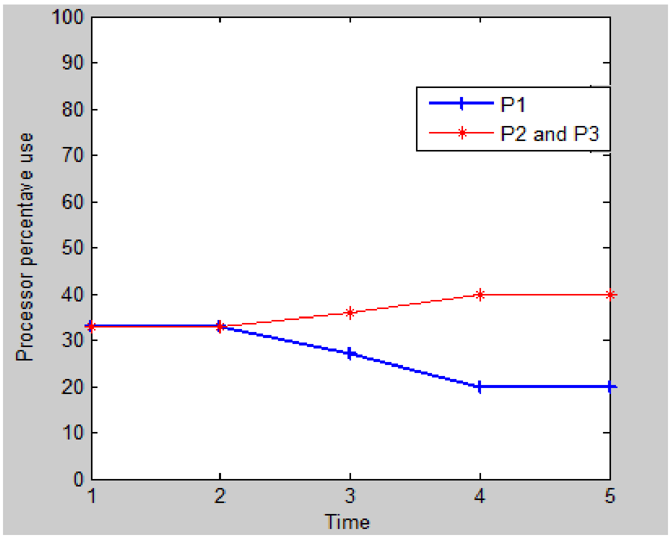


Figure 4.2: 20% Upper limit rule applied to applied to processor usage for process P1, competing against processes P2 and P3

It would be also possible that the resource allocation is reduced to less than 20% after a while in this example. While upper limit rules define a clear maximum resource allocation they not reserve a minimum resource allocation. Although if needed a process can have both lower an upper limits to maintain the resource allocation within a certain range. In figure 4.3 an example where process P1 is competing against processes P2 and P3, however both a lower limit of 20% and an upper limit of 40% are established to P1 during its entire execution.

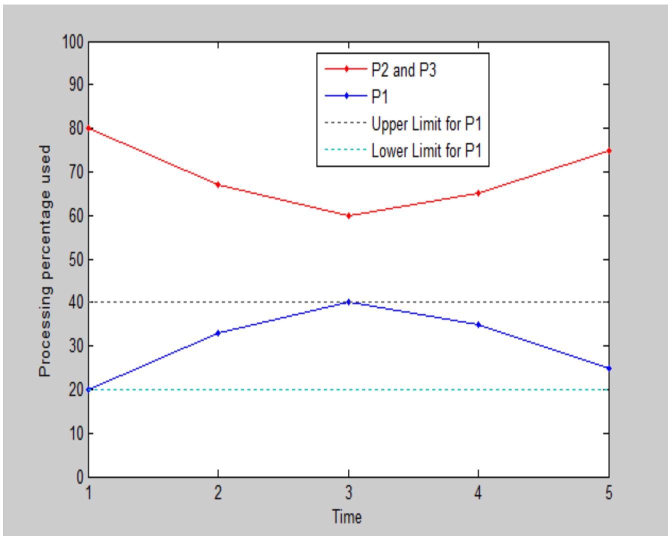


Figure 4.3: 20% Lower limit and 40% Upper Limit rules applied to applied to processor usage for process P1, competing against processes P2 and P3

In figures 4.1 and 4.2 the processor resource allocation took some time to be stabilized in 50% or 20%. This happens because the rule is being applied after the second cycle of time in these examples (rules can be applied dynamically), so they might take some time to take effect. That being said, a transition period to stabilize the process's resources allocation to meet allocation rules is expected, once a processes that has rule assigned to it is launched and its competing against other processes for resources.

For processes which limit rules weren't associated, the standard scheduling algorithms will be applied. Fairness in the resource distribution should be expected in these cases.

4.2 Discussion

In order to deal with a vast number of applications, most mobile devices' platform designers opted by an aggressive resource management strategy to increase system performance and throughput in general. This approach does not take into consideration what would be the correct mix of resources for specific usage cases. In a scenario with too many applications competing for resources, it is possible that this strategy would allow some low importance applications to consume a significant part of the resources available, impairing the execution of applications that needed better response times. By having critical applications response time affected, the overall security of the platform could be compromised or the user could experience undesired latency times when accessing his applications.

This chapter proposed alternatives to this resource management model. While it does not necessarily ensures the maximum system throughput, this work aims to ensure that non-critical applications get limits on resource consumption. Also it aims to ensure critical applications will always have a minimum of resources reserved, to meet the desired performance expected by the user, application requirements and platform security overall. The following chapter will discuss a proof of concept implementation to validate the model described in this chapter.

Chapter 5

Proof of Concept Implementation

This chapter discusses a simple proof of concept implementation of the resource management solution in proposed model. Due to the open source nature, resources availability and documentation this prototype was implemented in the Android platform. To prove the prototype viability, this implementation addressed the proposed limit rules, proposed in chapter 4, and it was compared against the current Android resource management system.

5.1 Implementation strategy

Benefiting from the Linux Kernel, the Android system schematic utilizes a unique user ID for each application. That allows to easily identify and segregate tasks and threads from specific applications. Taking advantage also of the native implementation of control groups in Android, this prototype relied on existing OS features to map and implement the proposed resource management model in the Android Platform.

To be able to map the proposed resource model into the Linux kernel, implementing the upper and lower limit resource managing rules, adjustments in the kernel layer were required. This implementation opted to bound groups of process from applications (that are ran through unique users IDs) to resources limits rules (upper or lower limits). Instead of adding complexity to this prototype, by modifying directly the resource manager scheduler, this works opted to adapt an existing implementation of Linux Control Groups used in Android.

The goal of this implementation, is to provide means to define rules that correspond to the new resource management model proposed on chapter 4. By adjusting this model to resources and collections of applications that can be controlled by Linux Control groups, a simpler, light weight, and easy to manage resource management model is available.

While this implementation is not adding new resource management capabilities to the kernel at first, it provides new means to future platforms to be able to define resource utilization limiting rules, increasing security, QoS, and attending the proposed resource management model.

During this work it was noted that Android already implemented a sort of implicit upper limit rule. By default, there is a fixed 95% limit of processing resources to foreground apps, and 5% for background apps and services. However, as previously discussed, this was designed to improve the user experience while using the interface, and to maximize the system throughput as a whole, not taking into consideration application requirements.

In this work the resource management limits were mapped using control groups for applications that require a differentiated resource scheduling. Lower limits were created by adding a new control group, that reserves processing resources to a specific application or a group of applications. This new group should be created on different hierarchy as the foreground apps cgroup. Note that a single task can be in multiple cgroups, as long as each of those control groups is in a different hierarchy. In other words, following this strategy cgroup creation strategy for lower limits, the applications were able to get resources from both the newly created custom group, that implemented the limit, and from the foreground, or background custom groups, that already existed in original operating system configuration.

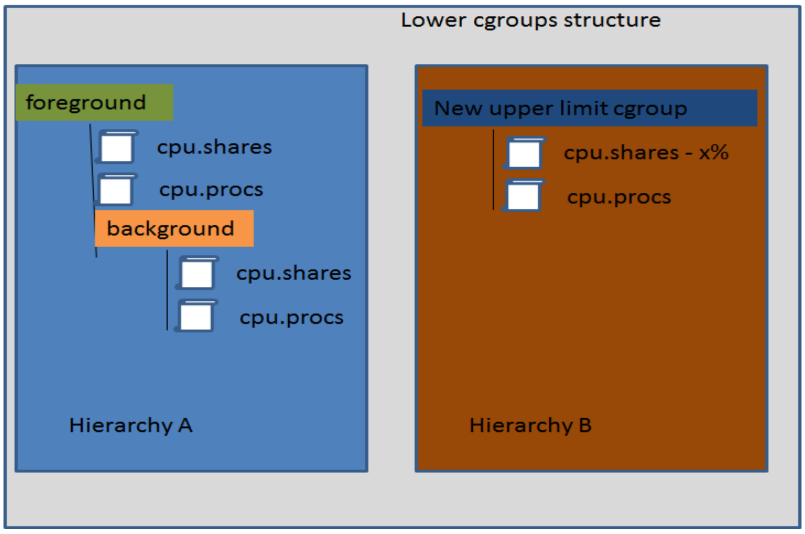


Figure 5.1: Representation of an lower limit implementation using cgroups

Upper Limits were created by implementing new control groups as well. There a two possible ways to do it. The option adopted in this implementation was to create this group within (one level below) the foreground control group, assuming that the foreground control group contain enough resources to attend the request maximum limit. As a task (process) can be added only to control group in the same hierarchy, there were no risks of gaining extra processor resources from the background or foreground custom groups. This happens because as soon as a task becomes a member of a second or third cgroup in the same hierarchy, it is removed from the first cgroup in that hierarchy. All tasks tasks are unable to be in two different cgroups in the same hierarchy at the same time.

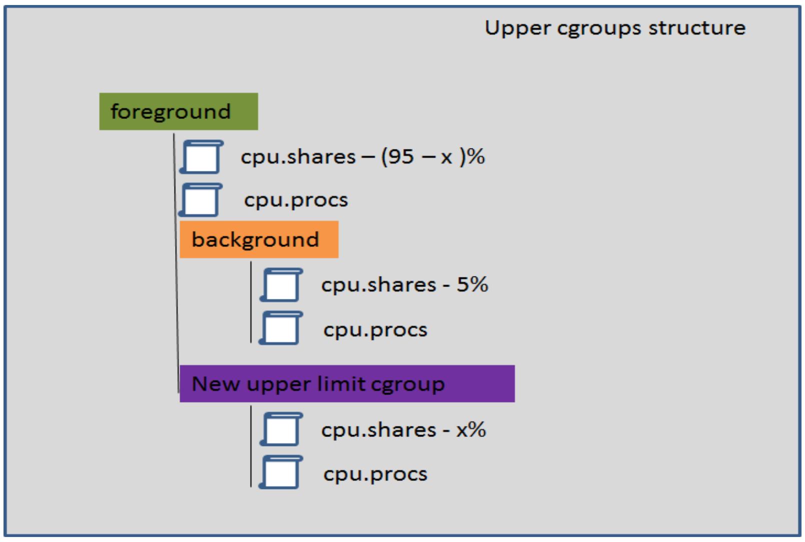


Figure 5.2: Representation of an upper limit implementation using cgroups

Another possibility would be to have a separated cgroup different hierarchic level from the foreground apps. This approach was not pursued at first in this work, due to the need to create an extra control to prevent tasks from being added into the foreground control groups structure, once the task was added to this new upper limit control group. This approach would increase the complexity of the prototype. Also it would limit the competition for resources against other applications.

It is known that poorly written rules could compromise the overall system performance, though this strategy provides a fine-grained resource control over the resources available to their applications. These rules are expected to be adjusted dynamically, by creating and adjusting custom groups on the fly while moving tasks through them.

The changes required into the existing control groups to implement the proposed model where performed directly in the Linux control group file system, via shell prompt, through ADB tool. ADB is a versatile command line tool that allow you communicate with USB connected Android-powered device or emulated devices, through a a client-server program [ADB, 2014].

Once a new limit rule was created the kernel file structure was updated, creating new control groups. Within this newly created group, files cpu.process and cpu.shares were populated. All processes created by the group of applications to whom the limiting rule was applied were listed on into file cpu.process. File cpu.shares was updated to contain the proportion of the resource assigned to the rule.

Due to the complexity of implementing a service to manage the control groups structure in real time the implementation of the cgroups structure was performed manually to each set up. It relied on working directly in the shell prompt, through ADB. It is expected that these shell commands would be standardized, and scripted in a platform that would really implement this model. These scripts would be part of a bigger piece, that somehow knows what the all applications' requirement are. When needed this bigger piece would implement the changes in the file system, as explained above, in order to implement the proposed limiting rules.

This implementation demonstrated the viability of the proposed model, using the Android platform. Due to being able to implement the proposed model using native Linux control groups, the overhead presented by this prototype was little to non-existent. This implementation is completely transparent to the end user. Unless limit rules were too poorly written, no difference should be noted in routine tasks.

5.2 Methodology and challenges

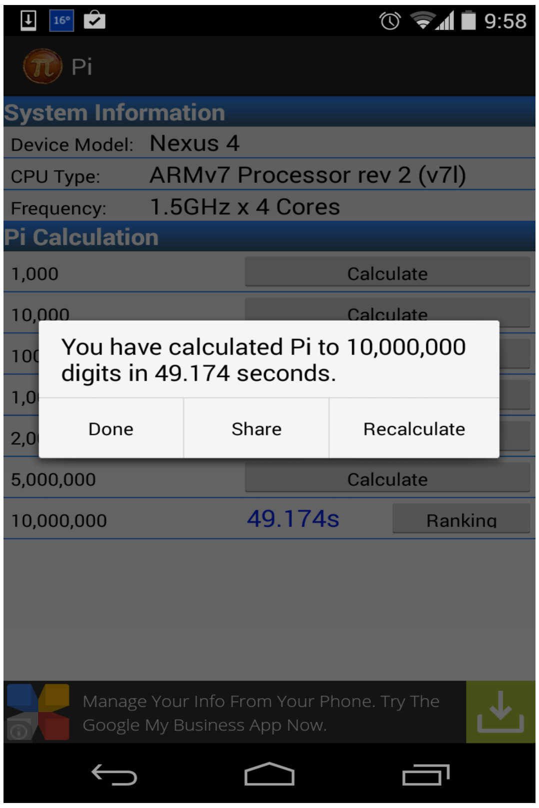
The development of this work and system tests were performed under a Google Nexus 4, running Android Kit Kat version 4.4.3. This device contains a Quad-core 1.5 GHz processor and 2 GB of RAM. To be able to touch the system kernel, the test mobile device had the OS rooted, following a similar process to the one described by [nexus4root.com, 2014]. It is noteworthy that rooting devices present some risks from bricking the mobile device to loosing warranty support.

At every round of tests a different setup was tested in this work, so the control groups file structure had to manually be updated by either creating new control groups or adjusting the schematic of existing ones.

In order to measure processor allocation command *top* was utilized, with huge amount of updates (-n) and very small wait time between them (-d). This was used to validate the success, when adjusting the upper limits for the foreground, and background control groups. To monitor specific processes of interest, launched by the group of controls to whom limiting rules where applied, the command | *grep* was added to improve the output. This command was redirected to text files for further analysis.

Several benchmark applications were downloaded to test the effects of custom groups adjustment that were performed, including CPU Benchmark [UnstableApps, 2014], Linpack for Android [GreeneComputing, 2014], Pi Benchemark [GGEEmulator, 2014], Quadrant Standard [Softworks, 2014] and Vellamo Mobile Benchmark [Qualcomm Innovation Center, 2014].

To better monitor the processor resource usage, and the focus exclusively on testing the processor in this implementation, the results evaluated were obtained applying limiting rules the to the Pi benchmark, an application that tests processing capacity by calculating π digits. On our tests π was calculated using 10,000,000 digits resolution. Tests were focused on running this application because it was the longer processor benchmark available, when considering elapsed time to run it. Tests were performed on both on the foreground and background control groups for each different limit rule set up. Depending on which test other programs, such as CPU Benchmark were run simultaneously in order to create competition for resources. Figure 5.3 shows the output produces by the PI benchmark.


9:58

Pi

System Information

Device Model: Nexus 4

CPU Type: ARMv7 Processor rev 2 (v7l)

Frequency: 1.5GHz x 4 Cores

Pi Calculation

1,000 Calculate

10,000 Calculate

100

1,0

2,0

You have calculated Pi to 10,000,000 digits in 49.174 seconds.

Done Share Recalculate

5,000,000 Calculate

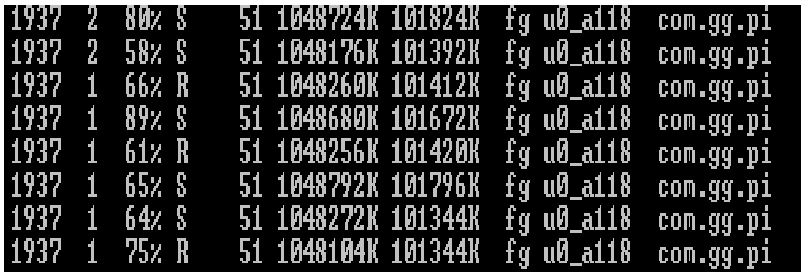
10,000,000 49.174s Ranking

Manage Your Info From Your Phone. Try The Google My Business App Now.

Figure 5.3: Snip of result when running PI benchmark in the foreground

During our testing both upper limit and lower limit rules were successfully implemented. It must be noted that upper limits are only applied once there applications competing for the resources. If the processor is not being used at a time and there is an application requesting for resources, this app will get the resource regardless of the upper limit. The upper limit will be implemented once another application claims these processing resources. On the other hand lower limits were always respected as long as the application is running and needing resources. Applications on pause state or not doing any process will not be claiming processor resources. As consequence of that the resources will be released to other applications.

It can be noted that even running in foreground the application does not necessarily uses 95% or more of the processing resource during the benchmark execution. Although during some tests the application reached the peak of 100 % of CPU utilization on the majority of the time of the samplings in this work the application did not require all processor resources or some of the resources where being used by other components on the foreground (e.g. notifications, notification bar, etc).



```
1937  2  80% S    51 1048724K 101824K  fg u0_a118  com.gg.pi
1937  2  58% S    51 1048176K 101392K  fg u0_a118  com.gg.pi
1937  1  66% R    51 1048260K 101412K  fg u0_a118  com.gg.pi
1937  1  89% S    51 1048680K 101672K  fg u0_a118  com.gg.pi
1937  1  61% R    51 1048256K 101420K  fg u0_a118  com.gg.pi
1937  1  65% S    51 1048792K 101796K  fg u0_a118  com.gg.pi
1937  1  64% S    51 1048272K 101344K  fg u0_a118  com.gg.pi
1937  1  75% R    51 1048104K 101344K  fg u0_a118  com.gg.pi
```

Figure 5.4: Snip of top command when running benchmark with a lower limit rule in place

5.2.1 Limit Rules Overhead and implementation time

As mentioned earlier no overhead was detected by implementing these limits using control groups. Also given the processor speed the transition period to apply a limiting rule could not be noticed as expected when the model was being created. The control groups structure were able to enforce the limiting rules instantly.

5.2.2 Results Discussion

To evaluate the prototype it was analyzed how long our different Limit rules set ups took to calculate the 10,000,000 π digits, while running the Pi benchmark. In table 5.1 several scenarios were conducted with different cgroups structures. Running on the foreground means that the benchmark was run in active state (activity running), visible on the main screen. Running in the background means that the application, was not visible in the main screen (running as a service). Columns *C. Apps F.* and *C. Apps B.* note if the process to which the limiting competing with applications in the foreground and background respectively, meaning that are other benchmarks instances were running simultaneously, competing for processing resources at the time of the test. If none of this column is Y (yes) it means that there are no other benchmarks competing for resources, just the OS services and notifications.

In round 1 it is established a foreground baseline to compare the Pi Benchmark. We use the standard Android OS configuration to run the tests, leaving the process running in the foreground cgroups.

In round 2, it is established a background baseline to compare the Pi Benchmark. We use the standard Android OS configuration to run the tests, however the benchmark is no longer running in the foreground cgroups, it is ran on the background one.

In round 3 it is established a background baseline to compare how the Pi Benchmark execution fairs against other benchmarks running in the foreground. Still using he standard Android OS configuration.

In round 4 it is established a background baseline to compare how the Pi Benchmark execution fairs when running against other benchmarks running in the background.

Table 5.1: Lower and Upper Limiting testing within Android Cgroups

Round	Average	Fastest	Slowest	Cgroup	C. Apps F.	C. Apps B.	50% UL	50%LL
1	48,929	42,853	55,013	Foreground	N	N	N	N
2	50,835	46,140	56,368	Background	N	N	N	N
3	85,710	72,853	96,785	Background	Y	N	N	N
4	92,680	86,903	102,408	Background	N	Y	N	N
5	59,439	54,580	64,221	Background	N	Y	N	Y
6	96,964	86,560	114,826	Background	N	Y	Y	N
7	61,311	57,894	64,202	Foreground	N	Y	Y	N
8	54,593	48,147	62,507	Foreground	N	Y	N	Y
9	51,716	48,597	55,683	Background	N	N	N	Y
10	52,964	48,333	58,313	Background	N	N	Y	N
11	49,359	44,321	54,285	Foreground	N	N	Y	N
12	50,104	44,688	55,277	Foreground	N	N	N	Y

In round 5 it can be noted the effect of having a lower limiting rule applied to our Pi Benchmark. We are no longer using the standard Android cgroups structure in this one. The PI benchmark is getting privileged resource assignment from the cgroup created to implement the lower limiting rule (50% of processing reserved) plus the processing scheduled by the existing standard background cgroup. Even though our application of interest was competing against other background process, like in round 4, the extra processing allocated by the lower limit rule resulted in faster execution times.

In round 6 we see the effect use an upper limiting rule of 50% applied to our Pi Benchmark. The results were similar to round 4, as expected. The new upper limiting cgroup prevented the resource allocation to go over 50% during the competition with other processes.

In round 7 the results were closer from the round 1 baseline. Although this might seem unexpected at first sight, this happened because the cgroups structure does not limit the resource allocation when there are no other applications claiming the resources. This assures maximum throughput to all applications when resources are not being utilized.

In round 8 again the results were closer from round 1 baseline, as expected. Applications got resources from both foreground and lower limit cgroups.

Rounds 9, 10, 11, and 12 validated the principle discussed in round 7. If the cgroups are in the same structure, it does limit the resources utilization, once there are no other processes competing for them, which can be perceived as a good thing, as maximize system throughput in general. These tables show that, regardless of having lower limits or not, the results were similar too baseline in the foreground in table round 1, that provided us a comparison point for the benchmark application running without resource competition.

To ensure that an upper limit was never exceeded regardless of the system having spare processing cycles or not it was needed to have the cgroups implemented in different hierarchy. The approach adopted to implement this during this implementation was to create the cgroups with a set of cores assigned to it. When implemented a 25% upper limiting rule through a cgroup with one core exclusively assigned to it, the scheduler ensured that the processes associated with the limiting rule always respected the limit of one quarter of maximum usage of the total process capability. On the other hand, this set of rule left this core idle once this application was no longer running, reducing the system throughput as a whole.

Chapter 6

Conclusion

6.1 Contributions

This work provided an overview about how resource managers work in mobile devices, and provided a deeper understanding of the functioning of the resource manager, and security features in the Android platform. It reviewed some of the state of the art resource management works and developed a new model for processor scheduling.

To the best of the author's knowledge this work inovates on defining a generic mobile device resource management model to the processor resources, based on the definition of upper an lower limits for resource assignments. This model can be used to contribute to the improvement of any mobile platform, as long as it works aligned with another piece that can properly define the applications' resource requirements.

It defined a resource management model meet the main goals by enabling mobile devices to implement new resource management rules, capable of privileging or limiting the scheduling of pre-determined processing resources to appropriate groups of applications. It also supported the creation of dynamic rules, that can be adjusted on fly, according to the system's needs. Lastly it also demonstrated how to perform a clean implementation of the proposed resource management model, through the use of cgroups in the Android platform, supporting this model's viability.

This implementation of this proposed resource management show that the resource limiting rules approach is viable to be implemented in future resource managers. Due to the fact that the implementation strategy adjusted components (cgroups) that already existed in the Android platform, virtually no overhead as added to the system.

6.2 Future work

Plenty of future work arose as possibilities of continuation of this work. This model could be adapted to control the resource scheduling resources such as memory, disk bandwidth and network. Implementations of similar resource managers for other resources in the Android platform could also take advantage of the native existing cgroups structure. This model could also be adapted to be applied in devices that are more resource constrained, and as result, generate guidelines to define resource limiting rules to such devices. Other mobile device's platforms, (Windows Phone, iOS, Symbian, etc.) could implement a similar resource management model to the one proposed in this work, evaluating them, and proposing specifics for other platforms.

Another possibility is to model usage control rules for other resources used by the applications, that were kept outside the scope of this work.

Mechanisms to define and evaluate applications' resource requirements can be created and work in conjunction with the proposed model. Ideally these new mechanisms would be able to adjust these requirements dynamically as the platform's applications change or evolve.

6.3 Conclusion

This work discussed some concerns related to mobile devices's security and resource management system. Reviewed cases where a differentiated resource management approach could increase Quality of Service and Security overall, by maximizing throughput for specific applications that could be critical to the system, through customized limiting resource management rules. Such ability becomes more important as mobile devices are becoming more targeted from attackers, and greater and newer critical applications are becoming more available for mobile devices.

It provided a new model, that can be applied to manage processing resources at the resource scheduler level, increasing the platform security and QoS. This model provides a theoretical basis to new resource managers, enabling the definition of upper and lower resource utilization limits, on the fly, once newer application requirements are in place. Such ability becomes more important everyday, as more and more work tasks is being performed through this class of devices.

While it is not expected to the average users to understanding and design resource management rules for their applications, this model supports the creation of a resource management mechanism that enables resource management prioritization, to certain applications. Ideally this piece would be part of a resource scheduler that automatically adjusts the resource allocation rules, in order to meet application requirements. How these application requirements are defined is out of the scope of this work.

This work also detailed how a clean resource management implementation can be implemented in Android platform, creating the proposed model through the usage of native Linux Control Groups. This approach was shown to be light-weight and viable. Through the usage of limiting lower and upper rules during the tests, application performance changed significantly confirming the functionality of the proposed model.

Assuming that application resource requirements are known, this new resource management model provide a viable strategy to meet application requirements in almost any platform, dynamically, on the fly.

Bibliography

[ADB, 2014] ADB (2014). Android debug bridge. retrieved from http://developer.android.com/tools/help/adb.html.

[AirWatch, 2012] AirWatch (2012). Enabling bring your own device (byod) in the enterprise. retrieved from http://www.air-watch.com/downloads/resources/byod-whitepaper.pdf.

[ANDROID DEVELOPERS, 2013] ANDROID DEVELOPERS (2013). Android developers documentation website. retrieved from developer.android.com.

[Apple, 2013] Apple (2013). Apple review guidelines. retrieved from https://developer.apple.com/appstore/guidelines.html.

[Braga et al., 2012] Braga, A. M., do Nascimento, E. N., da Palma, L. R., and Rosa, R. P. (2012). Introdução a segurança de dispositivos moveis modernos. pages 1–45, Curitiba,PR,Brazil.

[Chatterjee, 2013] Chatterjee, N. (2013). Designing efficient memory schedulers for future systems. Master's thesis, The University of Utah.

[Colmenares et al., 2013] Colmenares, J. A., Eads, G., Hofmeyr, S., Bird, S., Moretó, M., Chou, D., Gluzman, B., Roman, E., Bartolini, D. B., Mor, N., Asanović, K., and Kubiatowicz, J. D. (2013). Tessellation: refactoring the os around explicit resource containers with continuous adaptation. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 76:1–76:10, New York, NY, USA. ACM.

[Distimo, 2012] Distimo (2012). The battle for the most content and the emerging tablet market. retrieved from http://www.distimo.com.

[Enck et al., Feb] Enck, W., Ongtang, M., and McDaniel, P. (Jan.-Feb.). Understanding android security. *Security Privacy, IEEE*, 7(1):50–57.

[Felt et al., 2011a] Felt, A. P., Chin, E., Hanna, S., Song, D., and Wagner, D. (2011a). Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 627–638, New York, NY, USA. ACM.

[Felt et al., 2011b] Felt, A. P., Greenwood, K., and Wagner, D. (2011b). The effectiveness of application permissions. In *Proceedings of the 2nd USENIX conference on Web application development*, WebApps'11, pages 7–7, Berkeley, CA, USA. USENIX Association.

[Felt et al., 2011c] Felt, A. P., Hanna, S., Chin, E., Wang, H. J., and Moshchuk, E. (2011c). Permission re-delegation: Attacks and defenses. In *In 20th Usenix Security Symposium*.

[GGEEmulator, 2014] GGEEmulator (2014). Cpu benchmark. retrieved from https://play.google.com/store/apps.

[GreeneComputing, 2014] GreeneComputing (2014). Linpack for android. retrieved from https://play.google.com/store/apps.

[Jackson and Lameter, 2013] Jackson, P. and Lameter, C. (2013). Cgroups - manual. retrieved from https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt.

[Kim et al., 2011] Kim, H., Lee, M., Han, W., Lee, K., and Shin, I. (2011). Aciom: application characteristics-aware disk and network i/o management on android platform. In *Proceedings of the ninth ACM international conference on Embedded software*, EMSOFT '11, pages 49–58, New York, NY, USA. ACM.

[Lim et al., 2013] Lim, G., Min, C., and Eom, Y. I. (2013). Enhancing application performance by memory partitioning in android platforms. In *Consumer Electronics (ICCE), 2013 IEEE International Conference on*, pages 649–650.

[Muhammad Shiraz and Ahmed, 2012] Muhammad Shiraz, Abdullah Gani, R. H. K. and Ahmed, E. (2012). An extendable simulation framework for modeling application processing potentials of smart mobile devices for mobile cloud computing. *2012 10th International Conference on Frontiers of Information Technology*.

[Murmuria, 2010] Murmuria, R. (2010). Energy profiling & control for android devices. Master's thesis, George Mason University.

[Nauman and Khan, 2011] Nauman, M. and Khan, S. (2011). Design and implementation of a fine-grained resource usage model for the android platform. *Int. Arab J. Inf. Technol*, 4(1):440–448.

[nexus4root.com, 2014] nexus4root.com (2014). Nexus4root.com - nexus 4 root, custom roms, and more! retrieved from http://nexus4root.com.

[Oberheide and Jahanian, 2010] Oberheide, J. and Jahanian, F. (2010). When mobile is harder than fixed (and vice versa): demystifying security challenges in mobile environments. In

Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications, Hot-Mobile '10, pages 43–48, New York, NY, USA. ACM.

[OHA, 2013] OHA (2013). Open handset alliance members. retrieved from http://www.openhandsetalliance.com.

[Qualcomm Innovation Center, 2014] Qualcomm Innovation Center, I. (2014). Vellamo mobile benchmark. retrieved from https://play.google.com/store/apps.

[Russello et al., Oct] Russello, G., Crispo, B., Fernandes, E., and Zhauniarovich, Y. (Oct.). Yaase: Yet another android security extension. In *Privacy, security, risk and trust, 2011 IEEE third international conference on and 2011 IEEE third international conference on social computing*, pages 1033–1040.

[Schreckling et al., 2012] Schreckling, D., Posegga, J., and Hausknecht, D. (2012). Constroid: data-centric access control for android. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 1478–1485, New York, NY, USA. ACM.

[Shabtai et al., 2010] Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., Dolev, S., and Glezer, C. (2010). Google android: A comprehensive security assessment. *IEEE Security and Privacy*, 8(2):35–44.

[Softworks, 2014] Softworks, A. (2014). Quadrant standard. retrieved from https://play.google.com/store/apps.

[Starke, 2006] Starke, M. R. (2006). Controle dinâmico de recursos em sistemas operacionais. Master's thesis, Pontifícia Universidade Católica do Paraná.

[Thomson, 2012] Thomson, G. (2012). Byod: enabling the chaos. *Network Security*, pages 5–8.

[UnstableApps, 2014] UnstableApps (2014). Cpu benchmark. retrieved from https://play.google.com/store/apps.

[Vidas et al., 2011] Vidas, T., Votipka, D., and Christin, N. (2011). All your droid are belong to us: a survey of current android attacks. In *Proceedings of the 5th USENIX conference on Offensive technologies*, WOOT 11, pages 10–10, Berkeley, CA, USA. USENIX Association.