UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

COMUNIDAD ABIERTA EN EL DESARROLLO DE DISTRIBUCIONES BASADAS EN ANDROID

TESIS

Que para obtener el título de

INGENIERO EN COMPUTACIÓN

P R E S E N T A

JONATHAN DANIEL CRUZ RABADÁN

DIRECTOR DE TESIS

ING. JUAN J. CARREÓN GRANADOS

Ciudad Universitaria, Cd. Mx., 2018

AGRADECIMIENTOS

A la Facultad de Ingeniería de esta máxima casa de estudios de México, la UNAM, donde recibí la formación como ingeniero.

A la UNAM.

A mi asesor de tesis, el Ingeniero Juan José Carreón Granados, por permitirme desarrollar este trabajo bajo su asesoría.

A todos mis maestros de la carrera.

A Ira y Arturo, quienes siempre han demostrado ser padres ejemplares y me han apoyado incondicionalmente.

A mi hermano Saúl, porque es como mi segundo padre, y también el mejor hermano que alguien pudiera tener.

A mi abuela Carmelita (QEPD), a quien recuerdo con gran cariño.

A mis tíos, Victor y Nahúm.

A Mary Bell.

A mis amigos:

A Adrián, por escucharme, aconsejarme y ser un gran amigo para todo.

A Carlos, quien siempre me ha brindado gran apoyo tanto personal como profesional.

A Coco, Sergio, Humberto y Enrique, por haber estado en los momentos importantes de mi vida.

Y por último pero no menos importante, a Julio César, quien me enseñó que se puede volar en lugar de ser una persona gris.

Índice general

Resumen

Desde la introducción de Android como sistema operativo para teléfonos inteligentes, surgió una comunidad de desarrolladores independientes que se encargaron de personalizar el software de sus dispositivos, añadiendo características que al principio fueron consideradas faltantes en el software original y permitiendo a los usuarios configurar parámetros que de otra manera no hubieran podido modificar.

Con el tiempo, la comunidad abierta de desarrollo de software ha jugado un papel importante. Desde el inicio de Android, la comunidad abierta fue quien aprovechó una vulnerabilidad del sistema operativo presente en el HTC Dream que permitió al usuario adquirir permisos de súper usuario, iniciando así un camino donde aparecieron aplicaciones que Google, como principal desarrollador oficial de Android no tenía consideradas que existieran. Surgieron entonces aplicaciones que hacían uso de los permisos elevados concedidos, las cuales permitieron a la comunidad abierta ofrecer más libertad a los usuarios que eligieran modificar sus teléfonos.

La posibilidad de modificar el software fue percibido como una amenaza por parte de los fabricantes, quienes han ido perfeccionando mecanismos para impedir que los usuarios modifiquen el software original, y que en caso de hacerlo, el fabricante pueda invalidar de inmediato la garantía ofrecida a los consumidores o restringir el uso del dispositivo. Google, por otro lado, ha creado métodos de verificación que indican si el software en un móvil ha sido alterado, permitiendo así a los desarrolladores de apps usar esa información para impedir la ejecución de sus aplicaciones en ese dispositivo.

La modificación del Código Abierto de Android permitió que características que solamente se encontraban presentes en distribuciones no oficiales, fueran permeando a las oficiales, cerrando así la brecha entre las versiones originales y las modificadas.

Una posibilidad importante resultante del desarrollo de distribuciones de la comuni-

dad abierta es que permite tener una versión de Android reciente en dispositivos cuyos fabricantes abandonaron el soporte oficial de actualizaciones. Por lo tanto, podemos obtener un software actualizado si optamos por instalar una distribución no oficial en dispositivos antiguos pero que por sus características de hardware puedan ser utilizados de forma segura.

Abstract

Since the introduction of Android as an operating system for mobile phones, a new developers independent community was born, which was the one in charge of customizing the software running in their devices, adding features that were not included at the begining in the original software, and allowing users to change extra parameters which in other way they would not have been able to modify.

The open software development comunity has had an important role. Since Android's begining, that comunity was which took advantage of an operating system vulnerability that was found in the HTC Dream which permited users to gain root access. Since that, apps using root access have been developed, which allowed the open comunity to offer freedom to whom chose to modify his mobile's software, although Google had not considered it.

That new posibility was percibed like a security threat for makers, which have been improving mechanisms in order not to permit users modifying the original software, and whether it has been done, warranty might be void, or the functionality of the device would be restringed. On the other hand, Google has created methods of verification that indicate if the running software in the device has been modified, allowing app developers to use that information to block the execution of their apps in these devices.

Modifications over Android open source have allowed some features that were only present in no-official distributions arrived to the official ones, closing the gap between the original and the modified ones.

A quite important posibility when an open comunity distribution is ported to a device, is that the phone would have a newer operating system version even though the maker has laid it down as its official support period has finished. Thus, we may obtain an upgraded system whether we chose installing an open community distribution in

older devices as their specifications permit to be used securely for an extra time.

Capítulo 1

Antecedentes de Android

1.1. Introducción a los dispositivos móviles

1.1.1. Los dispositivos móviles

Desde el inicio del siglo XXI los dispositivos móviles han ido cambiando a la sociedad. Éstos nos permiten estar comunicados a cualquier instante, enterarnos de las noticias, el clima, conectarnos a los servicios que se ofrecen en la nube, obtener información del tráfico en las calles y no menos importante, los juegos.

Sus prestaciones han ido creciendo, ofreciendo características como memoria RAM,

CPUs y GPUs con rendimientos que hasta hace poco sólo se veían en dispositivos más grandes como las computadoras de escritorio y portátiles. Esta evolución no sólo se ha dado en el aspecto de hardware, sino también y de manera muy acelerada, en el software. De hecho, un dispositivo sin software no nos serviría para nada. En este tenor, el software que un usuario final ocuparía, consiste básicamente de un sistema operativo, y aplicaciones ejecutándose sobre éste.

1.1.2. Sistemas operativos para móviles

En los últimos 10 años han existido varios sistemas operativos para dispositivos móviles, principalmente en teléfonos inteligentes, entre los que destacan:

→ Android

→ BlackBerry OS

→ Firefox OS (B2G OS posteriormente)

→ iOS

→ Symbian

→ Tizen

→ Ubuntu Touch

→ webOS (Sucesor de Palm OS)

→ Windows Phone

Cada uno de ellos cuenta con una tasa de utilización diferente en el mercado, y el que domina actualmente es Android con un 74.39 %. En segundo lugar iOS cuenta con un 19.64 %, Windows Phone con 0.61 %, Firefox OS con 0.51 %, Symbian S40 con

0.50 % y Blackberry con 0.16 %, mientras que el 3.03 % restante pertenece a dispositivos desconocidos.[1]

Hasta el momento, varios proyectos han sido abandonados: Nokia lanzó la última versión de Symbian, renombrado comercialmente como Nokia Belle en octubre de 2012, cancelando futuros planes de actualizaciones. El soporte por parte de Nokia terminó el primer día de enero de 2014.

Firefox OS fue abandonado por Mozilla Foundation, y en septiembre de 2016 se anunció que su desarrollo no continuaría, retirando el código de Gecko, aunque el proyecto ha continuado por medio de forks, entre los cuales destacan el usado por Panasonic para sus Smart TVs que traían alguna versión de Firefox OS.

Blackberry OS dejó de ser desarrollado y en noviembre de 2013 se publicó la última actualización de Blackberry 10, la última versión lanzada.

Canonical cesó el desarrollo de Ubuntu Touch, aunque su desarrollo pasó a manos del proyecto UBportsm, el cual mantiene el desarrollo de dicho sistema operativo.

Windows Phone es otro de los sistemas operativos que ha fracasado al no atraer una cuota de mercado que le permita ser rentable y competitivo. En octubre de 2017 se confirmó que Microsoft no continuaría desarrollando nuevas características para Windows Phone, aunque seguirá proporcionando actualizaciones encaminadas a la seguridad por algún tiempo.

1.1.3. Estadísticas de uso de dispositivos móviles

En 2015 en los Estados Unidos, el porcentaje de usuarios exclusivos de Smartphones sobrepasó el porcentaje de usuarios exclusivos de equipos de escritorio, obteniendo en febrero de 2015 un 11.3 % para el primero y un 10.6 % para el segundo. De manera

[1] "Mobile Operating System Market Share Worldwide". Consultado el 25 de febrero de 2018 en `http://gs.statcounter.com/os-market-share/mobile/worldwide/`#monthly-201701-201801

general, hasta enero de 2018 había un 51.92 % de mercado para dispositivos móviles, un 43.87 % para dispositivos de escritorio y un 4.21 % para tabletas[2].

Durante 2017, Android encabezó la cuota de mercado a nivel mundial con un 69.68 % presente en móviles y tabletas, mientras que iOS acumuló un 23.44 %, dejando el restante para otras plataformas.

En México, los equipos desktop siguen dominando con un 59.3 %, mientras que los teléfonos móviles obtienen un 37.6 % y únicamente un 3 % para tabletas. En cuanto a los sistemas operativos, Windows (desktop) sigue a la delantera con un 45.88 %, Android con 32.39 %, OS X y iOS con 11.77 % y 7.12 % respectivamente. Linux obtiene una cuota de sólo 0.66 %.

1.2. Nacimiento de Android

Android Inc. fue una compañía fundada en octubre de 2003 en Palo Alto, California, EUA, que buscaba desarrollar un sistema operativo para utilizarlo en cámaras. Aunque el mercado de dispositivos móviles en ese entonces no era tan grande, pensaron en desviar el desarrollo del sistema operativo, enfocándose en uno para teléfonos inteligentes, que competiría con Symbian y Windows Mobile. Google adquirió a Android Inc. en 2005, después de que los fundadores, quienes habían estado desarrollando clandestinamente su sistema operativo se quedaran sin fondos.

El 5 de noviembre de 2007 Google liberó la primera versión beta de Android, y el 12 del mismo mes publicó la primera beta del SDK (Software Development Kit), pero no fue hasta el 23 de septiembre de 2008 cuando Google liberó la primera versión comercial de Android, la 1.0, también conocida como Apple Pie.

Las características básicas que destacaron sobre el diseño de Android, son:

[2]"Desktop vs Mobile vs Tablet Market Share Worldwide in January, 2018". Consultado el 28 de febrero de 2018 en `http://gs.statcounter.com`

→ Kernel basado en la versión 2.6 de Linux.

→ Las aplicaciones se ejecutan sobre una máquina virtual, una instancia por cada aplicación.

→ Las aplicaciones principalmente pueden ser escritas en Java.

→ Se concibe como una plataforma de software móvil estándar, de código abierto.

→ Se lanza como parte de la "Open Handset Alliance".

El primer teléfono móvil comercial en ejecutar Android fue el HTC Dream que fue lanzado el 22 de octubre de 2008, y contaba con la primera versión estable de Android, la 1.0, que corría sobre un hardware con un procesador de arquitectura ARM mono-núcleo a 528MHz y una RAM de apenas 192MB. La resolución de la pantalla era de 320x480 píxeles con 3.2 pulgadas de tamaño.

El sistema operativo tuvo tres actualizaciones disponibles para este HTC, llegando a la versión 1.6, llamada Donut, la cual integraba algunas características nuevas y una mayor estabilidad, aunque el limitado hardware se hacía ver en algunas situaciones.

Capítulo 2

Planteamiento del problema

Al ser Android un sistema operativo Open Source, puede ser descargado, modificado y utilizado por cualquiera. Android ha sido utilizado como sistema operativo por una gran variedad de fabricantes de dispositivos móviles, los cuales lo han utilizado desde hace varios años en teléfonos inteligentes, tabletas, televisores, relojes, y recientemente, también en automóviles, pudiendo añadir, modificar o dejarlo intacto al implementarlo en sus dispositivos, lo que le permitió ser aclamado como un sistema operativo muy versátil, y el más usado en dispositivos móviles a nivel global. Con el paso del tiempo y el crecimiento de la comunidad dedicada al desarrollo de Android, que en principio es desarrollado y mantenido por Google, se han desarrollado nuevos caminos para ex-

plotar sus características actuales, añadiéndole nuevas usando aplicaciones de terceros o desarrollando otras versiones basadas en AOSP.

La comunidad abierta también ha desarrollado sus propias distribuciones basadas en AOSP, modificando de acuerdo a diferentes necesidades el código liberado en cada versión por Google, y las cuales le han dado un nuevo soporte a cada vez más dispositivos móviles, enfocándose en teléfonos inteligentes y tabletas. Dichas modificaciones han traído consigo nuevas normas de implementación por parte de los fabricantes, dando o negando la posibilidad de que el software de sus productos pueda modificarse o sustituirse por completo.

Han surgido entonces varias preguntas sobre si es apropiado modificar el software de nuestros dispositivos móviles, principalmente por los temas de durabilidad, seguridad, y estabilidad, sobre todo al añadirse nuevas funciones que antes no se encontraban disponibles.

Por una parte, muchas distribuciones de la comunidad proporcionan versiones más nuevas del propio sistema operativo, permitiendo mejorar la seguridad contra ataques, disminuyendo los errores de seguridad que proporcionan las actualizaciones mensuales que son lanzados por Google y que son publicados en su boletín mensual. Aunque es cierto que al poder modificar el código, también es vulnerable a sufrir ataques por parte de los propios desarrolladores de la distribución, como la recolección de datos sensibles y sin autorización del usuario.

Por último, estas distribuciones han permitido extender el tiempo en el cual un dispositivo puede seguir recibiendo actualizaciones, incluso después de que el fabricante ha cesado de dar soporte.

2.1. Objetivo

Se buscará comprobar que las implementaciones de la comunidad informal basadas en el Sistema Operativo Android (AOSP) mejoran el rendimiento y alargan la vida útil de teléfonos basados Android al ofrecer frecuentemente una versión más reciente que la que fue desarrollada e incluida por el fabricante, y que puede ser una versión sin bloatware (Software inflado) y con diferentes características y funciones.

2.2. Método

Para el desarrollo de esta tesis se buscará información desde internet, principalmente foros, ya que es donde se concentra la información que un usuario puede consultar, y es donde la comunidad abierta publica sus distribuciones, así como tutoriales.

Se buscará una lista de distribuciones compatibles con un modelo específicamente: Motorola Nexus 6 (cuyo nombre código es "Shamu"). La fabricación de este modelo fue una colaboración entre Motorola y Google.

Se instalará alguna distribución que prometa una actualización importante al dispositivo, cuya versión de AOSP sea más nueva que la última ofrecida por el fabricante. Así se podrá tratar de comprobar si esta distribución ofrece ventajas al compararla con la última versión oficial. También se observarán desventajas y se podrá analizar si la relación entre ventajas y desventajas es favorable o no.

Capítulo 3

Android

Android es una pila de componentes open source que se ejecuta en una amplia variedad de dispositivos, en la mayoría de los casos, móviles. Esta pila es mostrada en la figura 3.1 donde se aprecia que consta de 5 capas.

En la capa inferior se encuentra el kernel de Linux, donde residen los drivers de los componentes como la pantalla, Bluetooth, y WiFi, entre otros. El kernel también se encarga de la administración de energía.

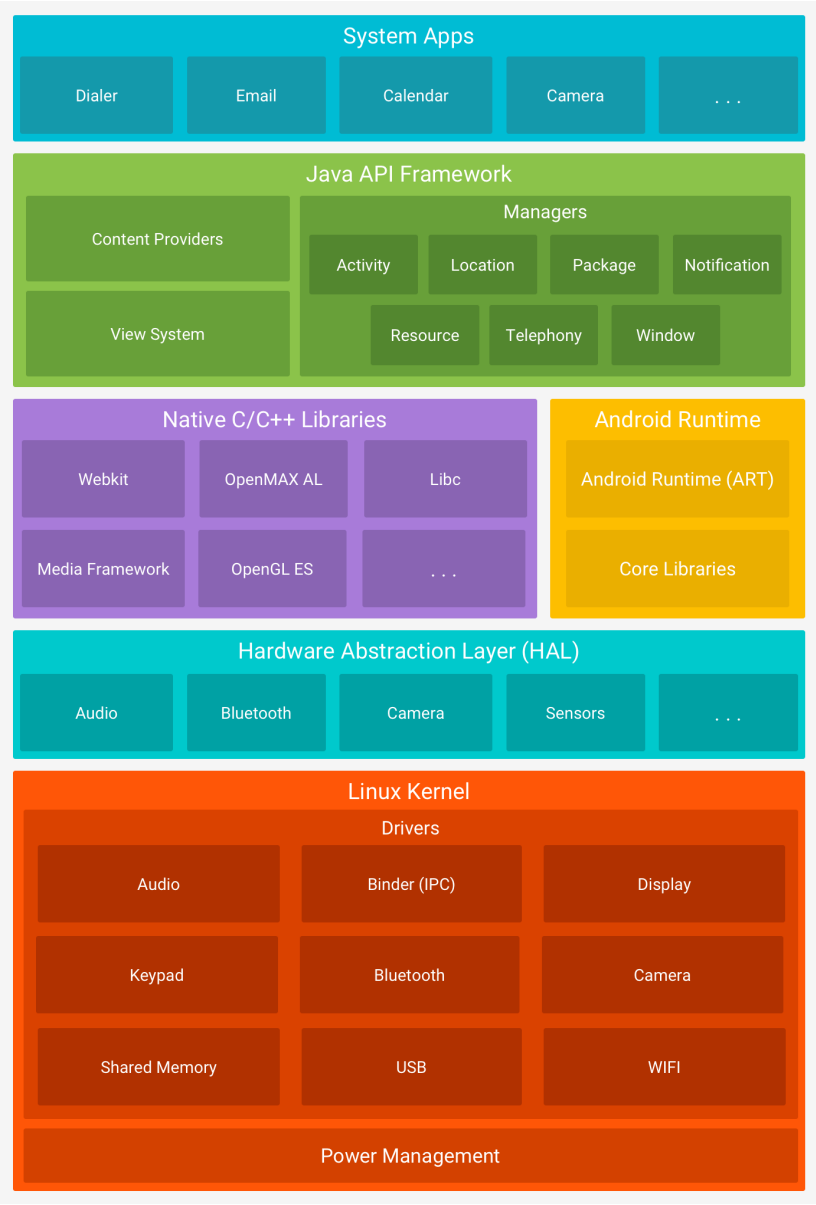

Figura 3.1: Pila de componentes de Android. Imagen tomada desde `https://source.android.com/devices/architecture/kernel/android-common`

El siguiente nivel es una capa de abstracción de hardware, HAL de sus siglas en inglés (Hardware Abstraction Layer), que consiste en librerías que son interfaces del hardware.

La tercera capa se encuentra dividida en dos:

Android Runtime (ART) o anteriormente Dalvik,

Librerías nativas de C y C++;

La penúltima capa es llamada Java API Framework, donde residen elementos como:

→ Content provider, que permite a las apps acceder a datos desde otras apps, o a compartir su propia información.

→ Notification Manager, que permite a las apps a mostrar alertas en la barra de estado de la pantalla.

→ View System, el cual puede ser utilizado para construir interfaces de usuario (UI).

→ Resource Manager, que proporciona acceso a recursos que no consisten en código, como imágenes.

→ Activity Manager, que administra el ciclo de vida de las apps en ejecución.

La capa superior corresponde a las apps del sistema, aunque pueden ser remplazadas por apps de terceros, como apps para mensajería, cámara, navegador de internet, etc.

3.1. AOSP

A.O.S.P. (Android Open Source Project) es el proyecto de Software *open source* que lidera Google, y al que cualquier persona u organización puede contribuir, y también utilizar para impulsar sus dispositivos. Se trata de un software diseñada para dispositivos móviles que está bajo ciertas licencias, como Apache Software License, Version 2.0. AOSP sólo es una parte del sistema operativo que se encontraría en un dispositivo móvil, que se complementará con software que podría no ser libre o abierto, como drivers para ciertos componentes, y por ello, el sistema operativo ya instalado en un dispositivo es una combinación entre código abierto, software libre y software privativo.

3.1.1. Versiones de Android por API

Para nombrar a las versiones del sistema operativo Android se utilizan las letras del abecedario y se elige el nombre en inglés de un postre. También internamente se designa un número de nivel de API (Application Programming Interface) pudiendo cambiar éste, sin cambiar el nombre de la versión. Cada API trae consigo nuevas características, tanto para el usuario final como para los desarrolladores.

La versión Donut es el nombre de la versión 1.6, seguida por la versión 2.0 Eclair, para luego continuar con la 2.1 Froyo, etc.

Donut fue publicada el 15 de septiembre de 2009, y su kernel se basó en la versión 2.6.29 del kernel de Linux. Su API level fue el 4.

Eclair se lanzó el 26 de octubre de 2009, y se basaba en el mismo kernel de la versión anterior, y su API level fue el 5. El 3 de diciembre se lanzó la versión 2.0.1, cuyo nombre no cambió por tratarse de una actualización menor, aunque su API level aumentó a 6. El 12 de enero de 2010 se publicó la versión 2.1 también llamada Eclair, con un API level 7.

El 20 de mayo se lanza la versión 2.2 cuyo nombre es Froyo, al cual le corresponde el API level 8. Froyo se basó en el kernel de Linux 2.6.32. De esta misma API se lanzaron tres actualizaciones, la 2.2.1, la 2.2.2 y la 2.2.3. Estas actualizaciones trataban sobre seguridad y cambios menores.

Las versiones con cambios más importantes han sido:

→ Android 2.3 “Gingerbread” API level 9.

→ Android 2.3.3 “Gingerbread” API level 10.

→ Android 3.0 “Honeycomb” API level 11, que únicamente fue concebido para tabletas.

→ Android 4.0 “Icecream Sandwich” API level 14, donde se integró la interfaz Holo de las versiones 3.x y que funciona tanto en teléfonos y tabletas.

→ Android 4.1 “Jelly Bean” API level 16.

→ Android 4.4 “Kitkat®” API level 19 y API level 20, esta última con soporte para dispositivos “wearables”.

→ Android 5.0 “Lollipop” API level 21. Se integra la interfaz “Material”.

→ Android 6.0 “Marshmallow” API level 23.

→ Android 7.0 “Nougat” API level 24.

→ Android 7.1 “Nougat” API level 25.

→ Android 8.0 “Oreo” API level 26.

→ Android 8.1 “Oreo” API level 27.

→ Android 9.0 “Pie” API level 28.

3.2. Máquina virtual

Android utiliza una máquina virtual en la que se ejecutan las aplicaciones. Dalvik fue la primera máquina virtual, que luego fue remplazada por ART (Android Runtime). Las aplicaciones son escritas en el lenguaje de programación Java, pero no se ocupa una máquina virtual de Java estándar para ejecutar las aplicaciones en Android, estableciendo diferencias de construcción.

3.2.1. Dalvik

Dalvik es la máquina virtual donde se ejecutaban las apps en las primeras versiones de Android, hasta la 4.4 Kitkat, y fue descontinuada en la versión 5.0 Lollipop. Con una herramienta como dx, el Java Codebyte era convertido en Dalvik's codebyte, lo cual era ejecutado por la máquina virtual. Por lo tanto, la máquina virtual Dalvik no es una máquina virtual de Java. A diferencia de la máquina virtual de Java, Dalvik fue perfeccionada para ejecutarse en dispositivos móviles, que poseen memoria limitada, con optimización en el uso de energía, y por lo tanto no poseía todas las librerías estándar de una máquina virtual ordinaria de Java; ejecutaba archivos .dex, además de ser una máquina virtual basada en registros, y no en una pila, como la máquina virtual de Java estándar.

3.2.2. ART

ART debutó en la versión 4.4 de Android, donde se podía elegir sobre Dalvik en los ajustes del desarrollador, como lo muestra la figura 3.2. ART es un tiempo de ejecución mejorado respecto a Dalvik, donde se mejoró la recolección de basura, optimizando el uso de memoria y fragmentación, y reducción del tiempo de recolección. Las aplicaciones son compiladas al bytecode de Dalvik y ejecutadas en ART. ART hace uso de una compilación AOT (Ahead-of-time), a diferencia de Dalvik, donde se utilizaba una compilación Just in time (JIT) la cual ocasionaba que el código debería ser compilado cada vez que se ejecutara una aplicación, que si bien no se compilaba todo el código de la aplicación sino las partes que son ejecutadas, ahora ART permite que la compilación del código se lleve a cabo en el momento de la instalación de la aplicación, trayendo consigo una instalación más lenta y un uso mayor de memoria, pero aumentando la velocidad en general de la ejecución de las aplicaciones.

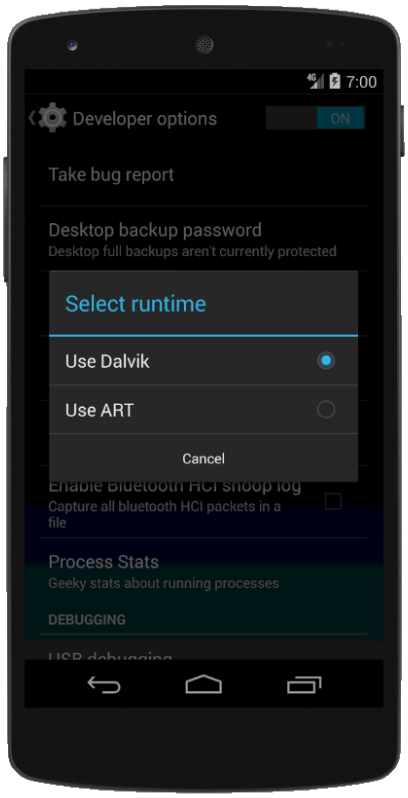

Figura 3.2: Máquinas virtuales disponibles en Kitkat.

3.3. Kernel

3.3.1. Kernel Linux

El kernel Linux es el núcleo de un sistema operativo libre, tipo UNIX, monolítico[1] y que soporta una gran portabilidad, como las computadoras personales, servidores, sistemas embebidos y dispositivos móviles, como Android.

El kernel se encuentra dividido en capas, principalmente en tres grandes: La interfaz de llamada del sistema, código del kernel independiente de arquitectura, y el código dependiente de la arquitectura.

El núcleo Linux se caracteriza por su gran eficiencia, sólida construcción y sirve

[1]Monolítico se refiere a que todos los servicios básicos son agrupados dentro del núcleo.

como ambiente de pruebas en la construcción y mejoramiento de nuevos protocolos. Se trata de un kernel dinámico, soportando la adición y la remoción de software *on the fly*. Quiere decir que pueden cargarse en todo momento módulos cuando sean necesarios.

Lanzamientos

Cada semana se lanza una versión del kernel, que a su vez es basada en un lanzamiento principal de Linus Torvalds. Estos lanzamientos semanales se tratan de versiones que no ofrecen un largo soporte en el tiempo, por lo cual desde la versión 2.6.16 se lanzó una versión estable y de largo plazo. Estas versiones LTS son seleccionadas aproximadamente una vez al año, ofreciendo un soporte mínimo de 2 años. Al día son publicados entre 6 a 8 parches de cada versión LTS, aunque dependiendo de factores, entre éstos destaca la edad del kernel, pues se vuelve más difícil aplicar parches a núcleos más antiguos debido a los cambios de código que el kernel va sufriendo, y muchos parches no son relevantes a los núcleos más antiguos, haciendo que mantener un kernel LTS se vuelva una tarea más complicada. Por lo mismo, existen ciertas reglas que deben seguir los parches que sean aplicados al kernel.

3.3.2. Kernel de Android

Android utiliza una versión del kernel Linux que proviene de las versiones LTS, e incluye parches de interés al desarrollo de AOSP que no han sido integrados a la versión LTS. Por lo tanto existen diferencias entre el kernel que ocupa alguna versión de Android, y la versión LTS del cual su kernel derivó, como las relacionadas a la arquitectura, manejo de energía, y uso del procesador.[2]

[2]Información extraída desde `https://source.android.com/devices/architecture/kernel/android-common` el día 15 de abril de 2018. Última edición el 9 de noviembre de 2017.

3.3.2.1. Requerimientos

Los kernels son puestos en una lista llamada AOSP common kernels, que debieron ser capaces de proveer un mecanismo para proveer al kernel de actualizaciones que incluyan los parches desde las actualizaciones de LTS, garantizar que el desarrollo de nuevas características no interfiera con la unión desde AOSP common, y un método para que los parches de seguridad desde el boletín de Seguridad de Android (ASB) sean fácilmente identificados.

Los Boletines de Seguridad de Android (Android Security Bulletins, ASB) son actualizaciones mensuales que tienen por objetivo mantener a los usuarios seguros. Las actualizaciones provienen de tres fuentes: desde AOSP, desde el desarrollo del kernel Linux y desde los fabricantes de SOC.

3.3.2.2. Ramas

En la figura 3.3, se muestra el procedimiento que se utiliza para mantener y actualizar el kernel en Android, en el caso particular de la versión 4.4.

Antes del lanzamiento de Android M NR1, android-4.4-n-mr1 es clonado desde android-4.4-n.

Únicamente los parches listados en ASBs son incluidos.

La siguiente versión android-4.4-n-mr2 será la versión anterior (android-4.4-n-mr1) más los parches desde el canal LTS que fueron incluidos entre ambos lanzamientos.

Cada mes cuando el ASB es dado a conocer públicamente, las ramas de lanzamiento son actualizadas en *upstream* con cualquier parche citado en el boletín.

3.3.2.3. Lineamientos

Las implementaciones de Android deben utilizar los siguientes lineamientos para el kernel:

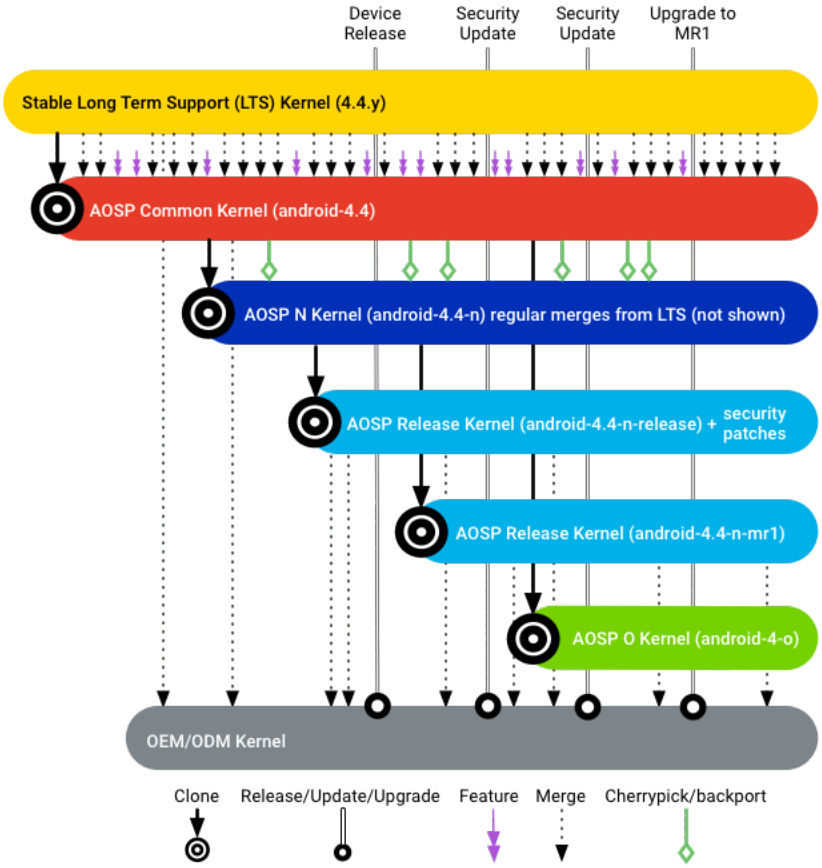

Figura 3.3: Kernel 4.4 branch. Figura tomada desde `https://source.android.com/devices/architecture/kernel/android-common`.

→ Usar los nuevos AOSP common kernels como una unión de fuente upstream.

→ Para obtener parches desde LTS, se combinará desde android-X.Y.

→ Se incluyen regularmente durante la fase de desarrollo.

→ Cuando se actualiza un dispositivo a una nueva versión, se combina desde la rama android-X.Y o desde la rama de lanzamiento objetivo.

→ Enviar correcciones a las versiones originales como LTS o a Android common.

3.3.2.4. Pruebas

En AOSP common kernels, pruebas regulares son efectuadas y los resultados están disponibles al público, específicamente:

→ Después de que son hechas actualizaciones desde LTS u otros parches son combinados, se realiza un test VTS (Vendor Test Suite) y un subconjunto de CTS (Compatibility Test Suite). Los resultados de dichas pruebas son publicadas en `https://qa-reports.linaro.org/lkft`.

→ Pruebas kernelci son realizadas con el objetivo de probar boots (arranques). Kernelci.org es un proyecto que prueba al kernel de Linux mediante test y validación automatizada de una amplia variedad plataformas que se ejecutan sobre Linux. Los resultados son publicados en `https://kernelci.org/job/android`.

3.3.3. Kernels personalizados

Los kernels personalizados son desarrollos que tienen como base el núcleo oficial del fabricante, pero con el fin de aportar nuevas características al kernel original de un dispositivo. Cuando se utiliza un kernel personalizado, es posible modificar parámetros como el gobernador del procesador (por ejemplo usar el gobernador on-demand en lugar de interactive), cambiar el voltaje en el rango de funcionamiento del procesador y desactivar cores, modificar las frecuencias tanto de CPU como del GPU, colores de la pantalla, añadir gestos por medio de tocar la pantalla, entre otras.

Algunos ejemplos de estos kernels son ElementalX, Franco kernel y LeanKernel.

3.4. Particiones estándar en Android

El almacenamiento en Android se encuentra organizado en diversas particiones, cada una con un propósito diferente. El número de particiones de un dispositivo es variable, de acuerdo al fabricante del dispositivo. Por lo tanto, el esquema tradicional MBR, que sólo permite 4 particiones es dejado de lado y en su lugar se utiliza una tabla de particiones GUID.

Se utilizan tanto tecnologías MTD (Memory Technology Devices) y eMMC (Embedded MultiMedia Card), aunque esta última es adecuada porque trabaja bien con sistemas de archivos basados en bloques, como el sistema ext4.

Android soporta varios sistemas de archivos, como FAT, exFAT, FAT32, NTFS, y los usados en Linux como ext4. F2FS es otro sistema de archivos que se ha utilizado en Android, y se encuentra en el kernel de Linux desde la versión 3.8.

La organización de las particiones es similar a un sistema Linux, pero existen diversas diferencias. El particionamiento genérico presente en un sistema Android con sus características, son las siguientes:

"/boot": Contiene el kernel y una imagen initramfs, que es usada para arrancar el sistema.

"/cache": Aquí se almacenan datos temporales.

"/recovery": Se almacena un kernel y un initframfs alternativo para iniciar una interfaz que permite la recuperación o restauración del sistema. Esta partición puede ser modificada remplazando la interfaz de restauración de fábrica por una personalizada (Custom recovery), que permitiría remplazar el sistema operativo, entre otras cosas.

"/userdata": Aquí son almacenados datos de configuración del usuario.

"/": Es el sistema de archivos raíz. Es montado desde initframfs. Aquí son montados los demás sistemas de archivos.

/**system:** Se trata de la carpeta donde Android despliega todos sus componentes. Por defecto, esta carpeta tiene permisos 755, pero es montado como sólo lectura, ya que brinda estabilidad y seguridad. Se encuentra organizado en subdirectorios.

Carpeta	**Descripción**
app	**Aplicaciones del sistema, como las provenientes de Google, del fabricante o del prestador de servicios de telefonía.**
bin	**Binarios, demonios y enlaces a comandos de shell. Ejecutables nativos de Android. Ejecutables nativos de Android.**
build.prop	**Propiedades en la construcción, usadas en el proceso de arranque.**
etc	**Archivos de configuración.**
fonts	**Archivos de fuentes .ttf**
framework	**Frameworks de Android, contiene archivos .jar y sus ejecutables .dex.**
lib	**Librerías .so**
lost+found	**Contiene operaciones de fsck.**
media	**Archivos de audio para la alarma, notificaciones, tonos de llamada, en formato .ogg.**
priv-app	**Aplicaciones privilegiadas**
usr	**Archivos de soporte, como plantillas de teclado.**
vendor	**Archivos específicos por el fabricante.**
xbin	**Binarios de uso especial.**

3.5. Bootloader

Bootloader, o también conocido como "Bootstrap", es el programa que se encarga de iniciar un sistema operativo cuando el dispositivo es encendido. Normalmente, el bootloader es ejecutado después de que el BIOS ha realizado tests como la comprobación de memoria y de hardware.

3.5.1. Bootloader en dispositivos Android

En móviles Android, también es el encargado de gestionar actualizaciones, iniciar en modo seguro, o en modo Recovery. Como se trata de una partición dentro del almacenamiento del dispositivo, puede ser modificada, por ejemplo, en una actualización.

Cada fabricante utiliza su propia implementación, por lo que no hay un estándar que regule el funcionamiento en todos los dispositivos, y no forma parte del código abierto de Android.

Un bootloader cualquiera debe proveer:

→ Soporte reducido del hardware,

→ Localizar e iniciar una bootimg, lo que permite pasar el control al kernel y empezar a usar un ramdisk,

→ Proporcionar una interfaz gráfica reducida,

→ Ofrecer una interfaz de comunicación con su host vía fastboot,

→ Proveer soporte de almacenamiento flash, que serviría en las actualizaciones,

→ Habilidad para manejar imágenes firmadas digitalmente mediante certificados SSL.

El bootloader se encuentra protegido (bloqueado) en todos los dispositivos Android como medida de seguridad. Únicamente son permitidas las actualizaciones que se encuentran firmadas digitalmente, usando algún certificado como X.509v3.

3.6. Root

En sistemas tipo Unix, *root* es el nombre del usuario administrador que tiene acceso a todos los archivos dentro de un sistema y a la ejecución de todos los comandos, a lo que se le denomina *root privileges* (privilegios de usuario root), y es la cuenta con más privilegios y por lo tanto, la que tiene el control absoluto del sistema.

El uso del término root para designar al usuario con permisos de administrador podría haber surgido desde el hecho que root es el único usuario que tiene permisos de escritura sobre el directorio raíz (del inglés root) en analogía al sistema de archivos cuya jerarquía de directorios ha sido diseñado como una estructura de árbol invertido donde todos ellos parten de un sólo directorio, como la raíz de un árbol[3].

El uso de un usuario administrador proporciona la capacidad de restringir el acceso a partes críticas del sistema y a directorios de otros usuarios del sistema, asegurando que ningún usuario sin conocimientos pueda corromper al sistema. Es asumido entonces que el usuario con privilegios root tiene total conocimiento sobre lo que hace dentro del sistema, permitiéndolo ser muy flexible cuando es correctamente usado.

3.6.1. Root en Android

Cuando Android fue lanzado, el propio sistema no contemplaba que el usuario tuviera que hacer uso de permisos de súper usuario como sucede en cualquier PC con algún sistema operativo tipo Unix. El 4 de noviembre de 2008 fue publicada la primer

[3]"Root Definition". Consultado el 24 de marzo de 2018 en `http://www.linfo.org/root.html`

guía en el sitio `forum.xda-developers.com` sobre cómo conseguir permisos elevados, específicamente para el dispositivo HTC Dream, detallando las ventajas de tener un teléfono con acceso root, entre las que destacaban la posibilidad de instalar alguna distribución de Linux, como Debian, y sustituir o mantener Android, entre otras. Dicha capacidad de obtener permisos de administrador se debieron a un fallo de seguridad presente en el firmware del teléfono. Las versiones RC 27 y RC 29 de dicho firmware eran vulnerables al fallo de seguridad, causando que el texto tecleado fuera reconocido como comandos con permisos elevados. De inmediato fue lanzada la versión R30 que corregía el fallo, aunque en la publicación del foro donde se detallaba el procedimiento para "rootear" el dispositivo, se actualizó la información y en un principio fue obligatorio mantener la versión sin actualizar a la versión R30, o hacer un *downgrade*[4] y luego realizar el procedimiento. A partir de entonces, muchas aplicaciones fueron desarrolladas específicamente para teléfonos previamente rooteados, desde aplicaciones que permitían encender la luz de la cámara y usarla como linterna, o para cambiar la partición donde se instalaban las aplicaciones, pues aunque el dispositivo contara con una segunda memoria de almacenamiento extraíble, Android en sus primeras versiones no contemplaba la posibilidad de usarla como memoria de aplicaciones, sino únicamente de almacenamiento secundario, donde se guardaban fotografías, vídeos y multimedia en general.

En general, se puede hablar de 4 formas de obtener permisos de súper usuario (Tyler, Jason. XDA Developers' Android TM Hacker's Toolkit. 2014. Página 31):

→ OEM software para instalar por medio de flash

→ Exploits

→ Flash nativo por Fastboot

[4]Volver a una versión más antigua de un software que la instalada actualmente.

→ Scripts o métodos automatizados

En esta tesis se usarán métodos automatizados por medio de aplicaciones de terceros.

3.6.2. Systemless root

Para que el demonio "su" permitiera obtener permisos de súper usuario, era necesario que iniciara su ejecución en el proceso de booteo. Desde Android 5.0 esto fue imposible, por lo que se introdujo el término *root systemless* en donde el acceso se obtenía con una imagen boot.img modificada. Por lo tanto, ya no se modifica la partición /system.

3.6.3. Seguridad relacionada al acceso root en Android

En general, el acceso a permisos de súper usuario en Android se obtiene al aprovechar alguna vulnerabilidad o *backdoor* en el sistema operativo, incluyendo al kernel. Es considerado que el acceso root en Android es un problema de seguridad en sí mismo, porque abre la posibilidad a que la información del usuario pueda ser leída por cualquier aplicación que ya haya ganado permisos de súper usuario, sumado a que para que esto sea posible, normalmente es necesario tener un bootloader desbloqueado.

Por lo tanto, se ha introducido Safetynet que permite a los desarrolladores elegir si su aplicación puede ser descargada, instalada y ejecutada en un dispositivo con acceso root. Un ejemplo es Android Pay.

3.7. Android Debug Bridge

Android Debug Bridge (adb) es una herramienta versátil de línea de comandos que permite la comunicación con una instancia del emulador o un dispositivo físico que ejecuta Android. Es un programa "Cliente-Servidor" que incluye tres componentes:

→ Un cliente, el cual se ejecuta en el equipo de desarrollo. Es posible invocar a un cliente desde una terminal usando un comando adb.

→ Un servidor, que se ejecuta como un proceso de fondo en la máquina de desarrollo.

→ Un demonio, que se ejecuta como un proceso de fondo en cada emulador o dispositivo físico. Adb puede ser encontrado en el directorio <sdk>/platform-tools.

Cuando se inicia una instancia adb, el cliente primero verifica si ya hay algún proceso servidor adb ejecutándose. Si no, éste inicia el proceso servidor. Cuando el servidor inicia, se enlaza al puerto local TCP 5037 y escucha por comandos enviados desde los clientes adb. Todos los clientes adb usarán el puerto 5037 para comunicarse con el servidor adb.

El servidor establecerá conexión con todos los emuladores o dispositivos. Encontrará emuladores o dispositivos buscando puertos impares en el rango 5555 a 5585, que son los utilizados por los emuladores o dispositivos. Cuando el servidor encuentra un demonio adb, establece una conexión a ese puerto. Antes se mencionó que cada instancia adquiere un par de puertos consecutivos. Por ejemplo, para la primera instancia, el puerto 5554 corresponde al puerto de consola, y el 5555 es el puerto adb.

3.8. SDK

Del acrónimo en inglés "Software Development Kit" se trata del conjunto de herramientas que nos ayudarán a desarrollar aplicaciones, en este caso para Android. De manera predeterminada, Android Studio no cuenta con todos los paquetes disponibles. Podemos personalizar los paquetes que utilizaremos si abrimos el gestor, hacemos clic en el icono como muestra la figura 3.4.

Al hacer clic en el ícono, aparecerá el "Android SDK Manager". Aquí podemos

Figura 3.4: Acceso al SDK Manager.

seleccionar las herramientas que vayamos a utilizar. Más abajo dentro de la misma lista se encuentran las APIs o versiones para las cuales queremos desarrollar, , como aparece en la figura 3.5. La última versión corresponde al API 28 o Android "P" que aún no es lanzado y está disponible para que los desarrolladores ejecuten sus aplicaciones y hagan uso de las nuevas características disponibles en el API.

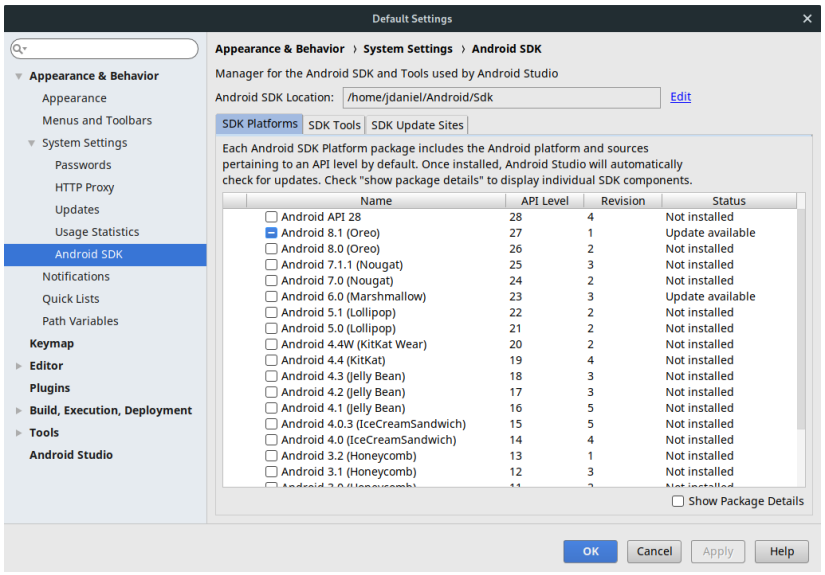

Figura 3.5: Android SDK Manager

De dicha lista, seleccionaremos el API 21 y el API 19 que corresponden a Android 5.0 "Lollipop" y a Android 4.4 "Kitkat" respectivamente.

3.9. Dispositivos virtuales

El emulador de Android es una aplicación que proporciona un dispositivo móvil virtual en en cual podemos ejecutar aplicaciones. El emulador corre una pila total del sistema Android, hasta el kernel, que incluye un conjunto de aplicaciones preinstalados

en las que podemos acceder desde nuestras aplicaciones. Podemos elegir la versión de Android a emular configurando “AVDs”.

El emulador de Android imita todas las características de hardware y software de un dispositivo móvil, con excepción de las llamadas telefónicas. Provee la capacidad de configurar, entre otros, la cantidad de la memoria RAM disponible y almacenamiento. Poseen una variedad de botones de control y navegación, así es posible generar eventos en las aplicaciones mediante clics del ratón o botones del teclado. También proporciona una pantalla configurable en diversos tamaños y resoluciones en la cual nuestras aplicaciones son desplegadas, junto con otras aplicaciones activas de Android.

3.9.1. AVD Manager

“Android Virtual Devices” es una herramienta con interfaz gráfica en la cual es posible crear y administrar dispositivos virtuales Android.

Para abrir el monitor de dispositivos, basta con hacer clic en el ícono de la barra de herramientas, como se muestra en la figura 3.6.

Figura 3.6: Android Virtual Devices.

A continuación aparecerá una ventana como en la figura 3.7, y que estarán listados nuestros dispositivos virtuales. Por defecto, está configurado el Nexus 5 con el API 21, el cual ya está instalado.

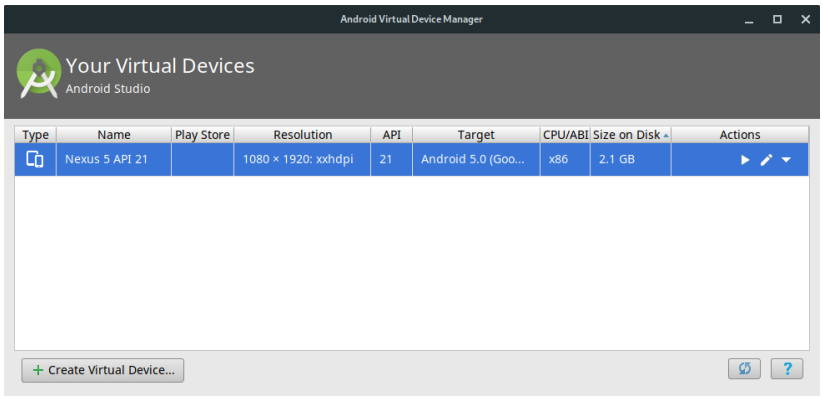

Figura 3.7: Android Virtual Devices.

Para iniciar el emulador, damos clic en el ícono verde que está en la columna "Actions". Para editar la configuración del mismo, hacemos clic en el ícono con forma de lápiz. A continuación en una nueva ventana como la de la figura 3.9, podemos cambiar parámetros como el tipo de dispositivo (ver figura 3.8), versión de Android, escala y orientación, y el rendimiento. En las opciones avanzadas, según la figura 3.10, se pueden cambiar parámetros más avanzados como la memoria RAM dedicada al emulador, la memoria "heap", la memoria de almacenamiento interno y la ubicación de un archivo que será la memoria SD. También podemos activar o desactivar las cámaras (frontal o trasera), teclado y la velocidad y latencia de la red. Las opciones "Use Host GPU" y "Store a snapshot for faster startup" no pueden ser activadas al mismo tiempo, la primera permite que el emulador ocupe las características de video del sistema en el cual se ejecuta el emulador (host) y la segunda opción guarda una imagen del emulador en un cierto estado de ejecución lo que permite un arranque más rápido y sin pérdida de configuraciones.

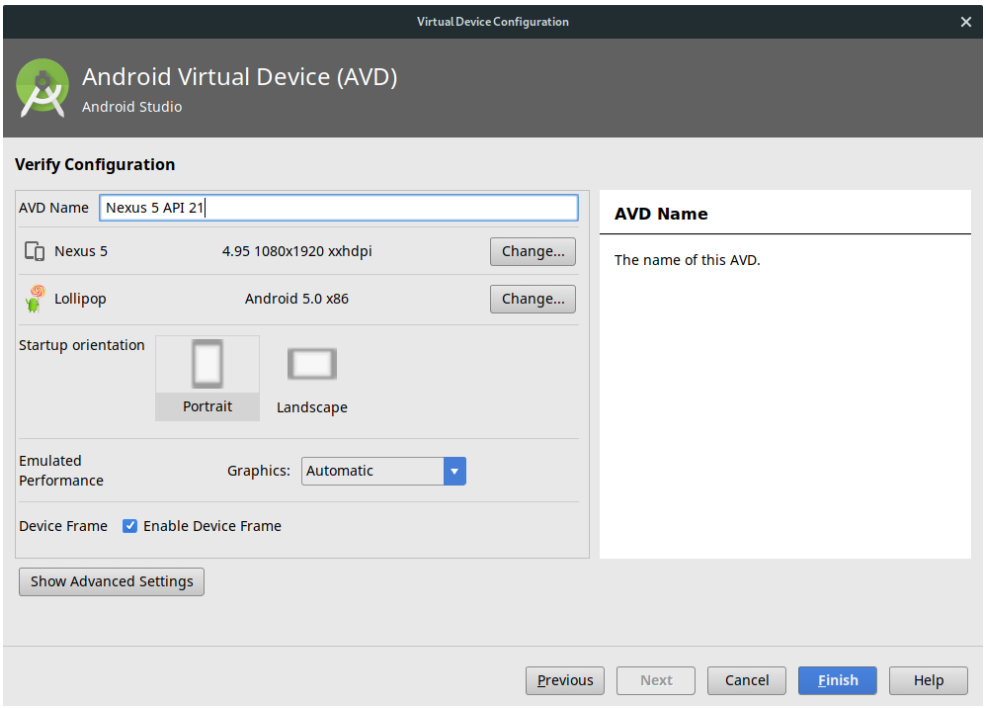

Figura 3.8: Configuración principal del emulador.

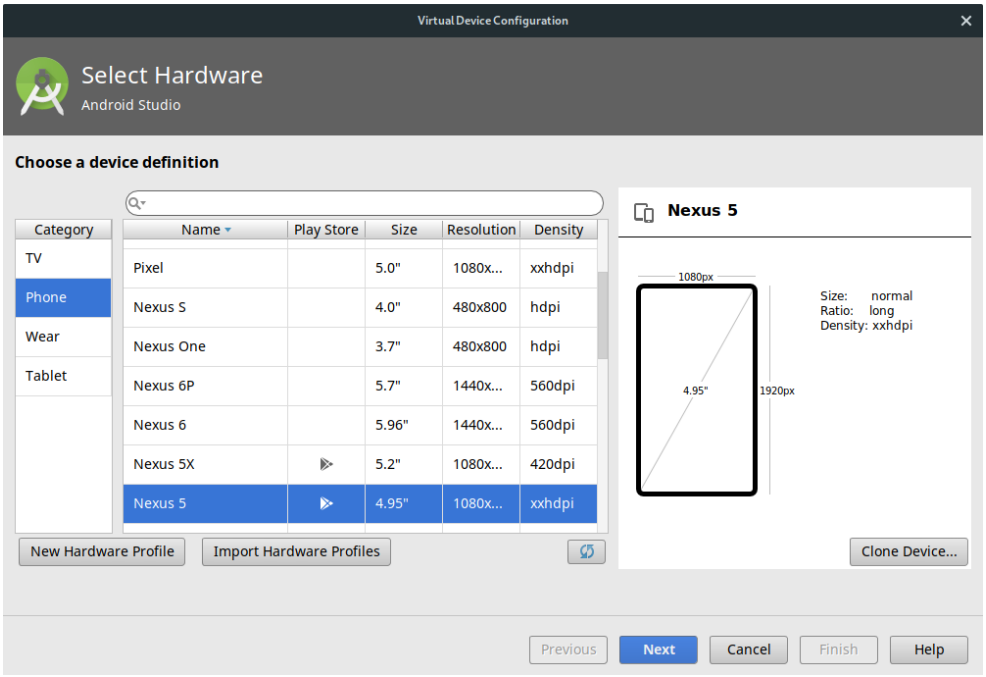

Figura 3.9: Tipo de dispositivo a emular.

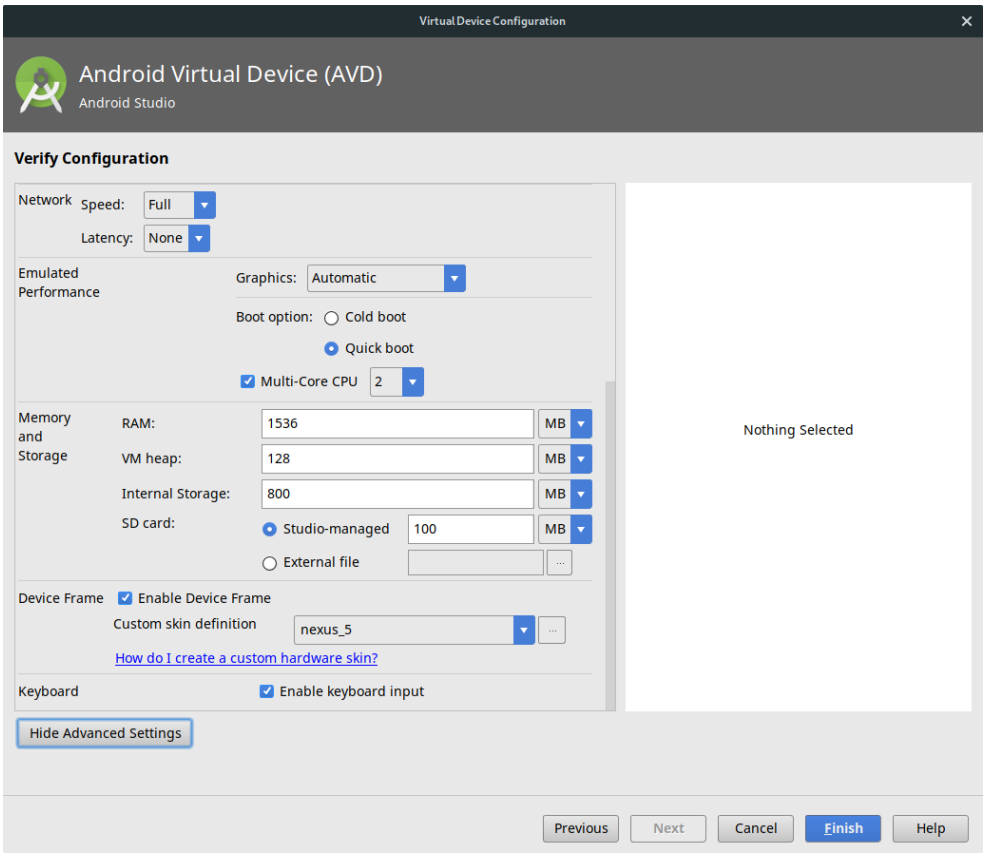

Figura 3.10: Configuración avanzada del emulador.

Después de finalizar de hacer los cambios, podemos iniciar el emulador. Esperamos algunos segundos a que se inicie para obtener en pantalla un emulador con Android 5 corriendo en él. De esta manera ya tenemos un dispositivo virtual en el que podemos hacer pruebas de nuestras aplicaciones en desarrollo. Algunas características son limitadas, como la autonomía de la batería, el flash de la cámara, etc.

3.9.2. Rendimiento de los emuladores

Muchos de los procesadores modernos proporcionan extensiones para ejecutar máquinas virtuales más eficientes. Tomando ventaja de estas extensiones con el emulador Android, se requieren algunas configuraciones adicionales pero mejoran de manera nota-

ble la velocidad de ejecución. Para determinar si nuestro sistema es capaz de aprovechar dichos tipos de aceleraciones, debemos verificar que el procesador del sistema cuenta con alguna de las siguientes tecnologías de virtualización:

→ Intel Virtualization Technology (VT, VT-x, vmx)

→ AMD Virtualization (AMD-V, SVM) (Sólo en Linux)

En Windows, Mac OS X y Linux es posible instalar el programa Intel Hardware Accelerated Execution Manager (HAXM). Éste permite mejorar el rendimiento de los emuladores por medio de asistencia de hardware, con la extensión de tecnología de virtualización Intel (Intel VT) utilizando imágenes de emulador Android x86.

En `https://software.intel.com/en-us/android/articles/intel-hardware-accelerated-execution-manager` están disponibles las guías de instalación de acuerdo al sistema operativo con el cual trabajamos. Para Linux, la guía se refiere a la distribución Ubuntu.

Para obtener aceleración para el emulador, no podremos ejecutar éste si nos encontramos dentro de una máquina virtual, como Virtualbox o VMWare. Para utilizar la aceleración con el emulador, necesitamos:

→ Android SDK Tools, revisión 17 o más reciente

→ Android basada en x86 (esta arquitectura no está disponible en todos los niveles de API).

La emulación de OpenGL no tendrá el mismo rendimiento que el proporcionado en un dispositivo real.

3.9.3. Comunicación con los emuladores

El emulador proporciona capacidades de red versátiles para configurar y modelar complejos ambientes para probar las aplicaciones. Cada instancia del emulador se ejecuta detrás de un servicio "router/firewall" virtual que lo aisla de las interfaces de red y configuraciones de internet de nuestro sistema. Un dispositivo emulado no puede ver nuestra computadora u otras instancias en la red. Únicamente ve que está conectado a través de ethernet a un router/firewall.

Como el emulador no se conecta directamente por hardware, éste y las aplicaciones se encontrarán con limitaciones:

→ La comunicación con el emulador puede ser bloqueada por un firewall ejecutado en nuestro sistema, o físicamente por un router al cual nuestro sistema esté conectado. El router virtual del emulador debe ser capaz de manejar todas las conexiones TCP y UDP en nombre del dispositivo emulado, siempre que el entorno de red de nuestro equipo lo permita. No hay límites en los rangos de números de puertos, excepto los impuestos por nuestro sistema operativo y red.

→ Dependiendo del ambiente, el emulador podría no ser capaz de soportar otros protocolos.

Para comunicarnos con un emulador detrás de su router virtual, es necesario configurar una redirección de red en el router virtual. Entonces los clientes pueden conectarse a un puerto invitado en el router, mientras el router dirige el tráfico hacia/desde ese puerto al puerto host del dispositivo emulado.

Hay dos caminos para configurar la redirección de red:

→ Usando la consola de comandos del emulador

→ Usando la herramienta ADB (Andorid Debug Bridge)

La redirección es útil cuando queremos conectar una instancia del emulador con otra, es decir, si nuestro equipo es A, una instancia del emulador es B y la otra instancia es C, y queremos que C se conecte a un servidor que está corriendo en B.

3.9.3.1. Comunicación al dispositivo virtual por medio de Telnet

Cada instancia del emulador que se está ejecutando proporciona una consola que nos deja consultar y controlar el ambiente del emulador. Podemos usar la consola para administrar la redirección de puertos, características de red y eventos de telefonía mientras nuestras aplicaciones se ejecutan en el emulador.

Para acceder a la consola de cualquier emulador en cualquier instante, usaremos telnet (con un emulador ejecutándose):

```
$ telnet localhost <puerto>
```

Cada emulador ucupa un par de puertos adyacentes: un puerto de consola y un puerto adb. Los números de puerto difieren por 1, y el puerto adb es el número de puerto más alto. La consola del primer emulador ejecutándose en una máquina usa el puerto de consola 5554 y un puerto adb 5555.

Para hacer conexión con un emulador, es necesario especificar un puerto de consola válido. Es posible encontrar el puerto del emulador al cual queremos realizar conexión, viendo el título de la ventana de éste, como lo muestra la figura 3.10. Para este ejemplo, el comando

```
$ telnet localhost 5554
```

establecerá una conexión con el emulador con el puerto de consola 5554. Una vez que se establece la conexión con el emulador, el comando “help” desplegará una lista de posibles comandos.

Figura 3.11: El puerto de este emulador es 5554, según la parte superior de la ventana.

Para ver el estado de energía del dispositivo virtual, escribiremos el comando "power" junto con el argumento "display"

```
power display
```

y la salida a este comando será

```
power display AC: online
status: Charging
health: Good
present: true
capacity: 50
```

Para cambiar el estado de carga del emulador a off/on

```
power ac off
```

```
power ac on
```

Si queremos cambiar el estado de la batería, a descargando

```
power status discharging
```

También es posible cambiar el estado de salud de la batería, indicando un sobrevoltaje

```
power health overvoltage
```

Y para cambiar el porcentaje de carga de la batería, usando un número del 0 al 100, en este caso al 85 %.

```
power capacity 85
```

Es posible simular llamadas y mensajes de texto entrantes y salientes usando telnet, o dos o más instancias del dispositivo virtual. Primero se mostrará cómo hacerlo con una sola instancia del emulador y telnet.

Para recibir una llamada en el dispositivo virtual, escribiremos el comando "gsm" seguido del subcomando "call", acompañado por un número telefónico cualquiera, por ejemplo

```
gsm call 550001
```

por lo que en nuestro dispositivo virtual recibiremos una llamada del número 55-0001, así como en la figuras 3.12 y 3.13.

Figura 3.12: Recibiendo una llamada generada en telnet del número 55-0001.

Para simular un mensaje de texto, escribiremos el comando "sms send <númeroEnvíaMensaje> <texto>"

```
sms send 550001 Hola, desde telnet
```

El mensaje recibido se muestra en la figura 3.14.

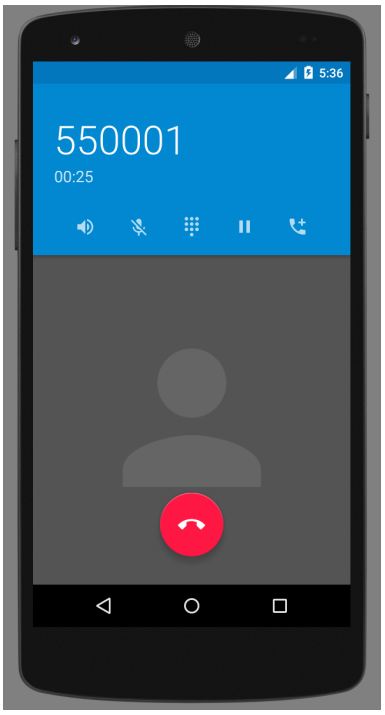

Figura 3.13: Aceptando la llamada generada desde telnet.

3.9.3.2. Comunicación entre múltiples instancias del dispositivo virtual

Es posible ejecutar varias instancias del emulador, ya sea configurando un nuevo dispositivo con características y un API de Android diferente, o iniciando otro desde la misma configuración. Como fue visto anteriormente, al configurar el dispositivo virtual en "AVD", iniciaremos otra instancia que consiste en el Nexus 4 con el API 17 (Android Jelly Bean 4.2.2) al mismo tiempo que está ejecutándose nuestra primera instancia que consta del Nexus 5 con API 22 (Android Lollipop 5.1).

Recordemos, la primera instancia del emulador, el Nexus 5, le corresponde el puerto 5554, y a la segunda instancia, el Nexus 4, le corresponde el puerto 5556. Realizaremos una llamada telefónica de la instancia con puerto 5554 a la 5556. Para esto, en la primera instancia iremos a la aplicación de llamadas y marcaremos el número correspondiente al puerto del dispositivo al cual queremos llamar, en este caso 5556, como lo muestra la

Figura 3.14: SMS recibido, generado desde telnet.

figura 3.15. Del lado derecho de la figura, el dispositivo 5556 se encontrará recibiendo la llamada desde 5554.

De la misma manera, es posible enviar SMS desde una instancia a la otra, o interactuar con las aplicaciones que nos encontremos desarrollando que requieran de varias instancias para hacer pruebas de funcionamiento.

3.10. Dispositivos Nexus

La línea Nexus fue una gama principalmente de teléfonos móviles, desarrollada por Google, la cual encargaba a algún fabricante para que éstos se encargaran del proceso de fabricación. Los dispositivos Nexus se beneficiaban de tener actualizaciones del sistema operativo Android prácticamente el mismo día del lanzamiento de alguna nueva versión o el lanzamiento de parches de seguridad, además de contar con un sistema operativo

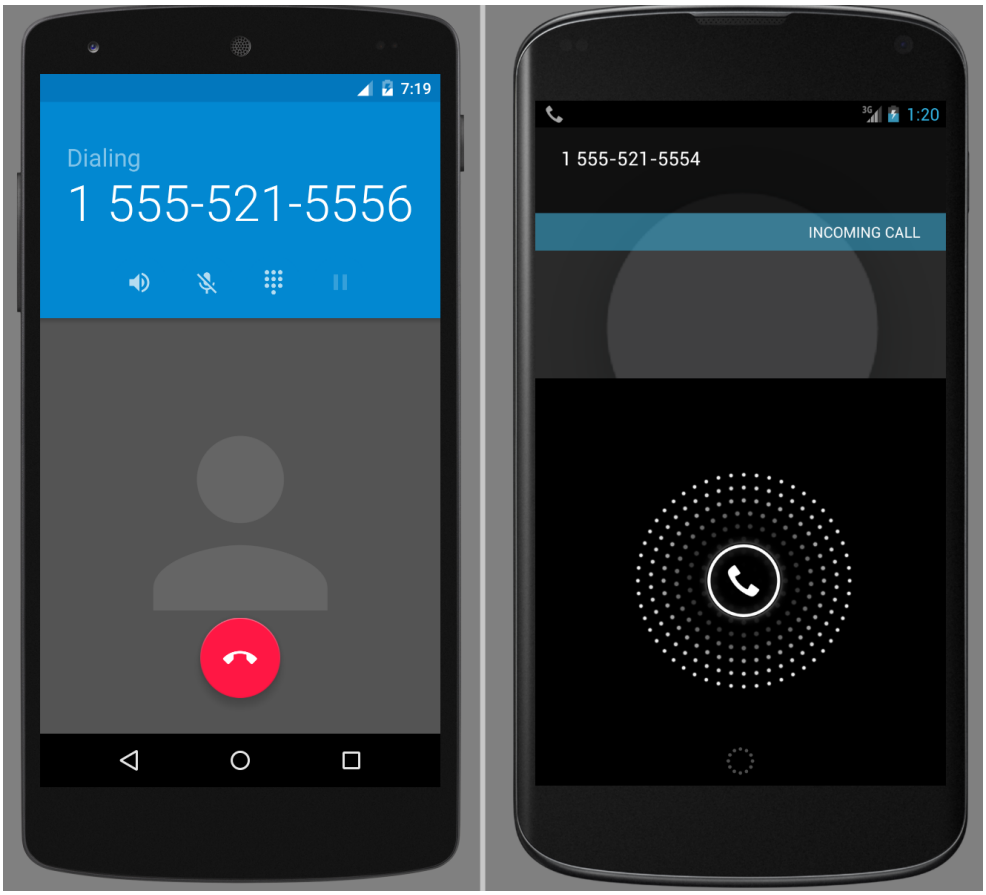

Figura 3.15: Llamada desde una instancia de emulador a otra.

"puro", lo cual significa que no existe ninguna capa de personalización o modificación alguna al sistema operativo Android que se ejecuta en el dispositivo, aunque eran dotados de las aplicaciones y servicios de Google. La línea Nexus fue descontinuada después del lanzamiento del Nexus 6P de Huawei y el Nexus 5X de LG, apareciendo en su relevo la línea de móviles Pixel. A continuación encontramos los dispositivos más importantes de la lína Nexus con sus características principales y fabricante:

→ Nexus One

Lanzado el 5 de enero de 2010. Contaba con Android 2.1 Eclair.

Poseía un CPU mono-núcleo Qualcomm Scorpion con 512 MB de RAM.

Fabricante: HTC.

→ Nexus S

Lanzado el 16 de diciembre de 2010. Traía Android 2.3 Gingerbread.

Era impulsado por un CPU mono-núcleo y una memoria compartida de 512 en RAM.

Fabricante: HTC.

→ Galaxy Nexus

Fue lanzado el 17 de noviembre de 2011. Traía Android 4.0 IceCream Sandwich.

Contaba con un procesador dual-core ARM Cortex-A9 con 1GB RAM.

Fabricante: Samsung.

→ Nexus 4

Fue lanzado el primero de noviembre de 2012. Ejecutaba Android 4.2 Jelly Bean.

Usaba un CPU quad-core Krait y 2 GB en RAM.

Fabricante: LG Electronics.

→ Nexus 5

Lanzado en octubre 31 de 2013. Corría con Android 4.4 Kitkat.

Tenía un CPU quad-core a 2.26GHz, con 2 GB en RAM.

Fabricado por LG Electronics.

→ Nexus 6

Fue lanzado en noviembre de 2014. Protagonizó con Android 5.0 Lollipop

Usa un procesador quad-core a 2.65GHz y tenía 3 GB en RAM.

Fue fabricado por Motorola.

→ Nexus 6P

Lanzado el 29 de septiembre de 2015. Traía Android 6.0 Marshmallow.

Contaba con un sistema de ocho cores (cuatro A53 y cuatro A57) y 3 GB en RAM.

Fabricado por Huawei.

3.11. Las ROM

Existen proyectos de distribuciones de Android de la comunidad abierta enfocados en uno o más dispositivos cuyo sistema operativo de fábrica es Android. Una distribución es llamada ROM por la comunidad abierta. Las ROM, que se basan ya sea en alguna versión de AOSP o en la versión de Android que el fabricante ha modificado para personalizarlo, el kernel y las aplicaciones del sistema, son el sustituto al sistema operativo original.

Es posible clasificar a los proyectos y sus ROMs según sean:

→ basadas en AOSP

→ basadas en el software Stock del fabricante.

Desde la aparición de Android, han existido muchos proyectos para crear y mantener estas distribuciones. El objetivo de cada proyecto no siempre es el mismo.

Cuando una ROM se basa en AOSP, se ofrece una experiencia de usuario de Android pura, pero añade nuevas funciones y características, además de basarse en las actualizaciones mensuales que Google libera. Estas distribuciones son las más comunes ya que no incluyen software propietario, incluídas la tienda de aplicaciones de Google o el cliente de correo de GMAIL, aunque estas aplicaciones pueden ser instaladas más adelante.

Las ROM basadas en el software stock de fábrica permiten portar versiones del software de otros modelos más nuevos a dispositivos antiguos, usando la capa de personalización del fabricante conservando las aplicaciones de fábrica. Las existentes para

dispositivos Samsung basadas en Touchwiz son un ejemplo, permitiendo actualizar, aunque en menor medida, dispositivos de la misma marca.

3.11.1. Distribuciones basadas en AOSP

Las ROM basadas en AOSP tuvieron una buena aceptación por parte de los usuarios ya que han permitido deshacerse de la capa de personalización de los fabricantes y de aplicaciones consideradas bloatware por parte del fabricante o por parte del operador de telefonía, y que frecuentemente no era posible eliminarlas.

3.11.1.1. CyanogenMod

Durante muchos años, Cyanogenmod (/saɪˈænoʊˌdʒɛnmɒd/) fue el proyecto de ROM de remplazo para dispositivos con Android más grande, ya que soportaba un gran número de terminales, y por lo tanto su equipo de desarrollo era amplio.

El fundador del proyecto fue Steve Kondik, quien usaba como alias en internet "Cyanogen". El equipo de trabajo creció convirtiéndose rápidamente en una distribución disponible para casi todos los dispositivos, teniendo 4 tipos de lanzamientos: nightly, milestone, experimental y stable.

A cada versión de Android le correspondía una versión de Cyanogenmod. Por ejemplo, Cyanogenmod 5 fue basado en Android 2.0, así como Cyanogenmod 14.1 es basado en Android 7.1.

Normalmente una nueva versión de Cyanogenmod aparecía unas semanas después del lanzamiento de cada versión de Android. El proyecto cesó en diciembre de 2016, aunque fue creado un fork llamado LineageOS que continuó con el desarrollo.

Ofrecía dentro de la rom aplicaciones desarrolladas por el equipo, como un lanzador de aplicaciones llamado Trebuchet, una lámpara llamada Torch, un gestor de archivos, aplicaciones personalizadas de teléfono y mensajes, un cliente de correo electrónico, una

aplicación de música, una aplicación para usar la cámara. Además, ofrecía un Recovery propio, compatible con sus propias distribuciones.

3.11.1.2. LineageOS

Se trata del proyecto que ha dado continuidad a CyanogenMod. Sigue siendo un proyecto que se caracteriza por ofrecer estabilidad, una versión actualizada de Android, y algunas características adicionales a las encontradas en AOSP.

3.11.1.3. SlimRoms

Slimroms es un proyecto que se basa en la promesa de tener una distribución de Android rápida y ligera, además de actualizada. Aunque no ha sido tan importante como Cyanogenmod, soporta varios dispositivos, aunque no ha tenido un desarrollo muy dinámico, pues los dispositivos no reciben actualizaciones tan rápido. La última versión para el Motorola Nexus 6 fue una versión de su canal beta del día 23 de junio de 2017, basada en Android 7.1.2.

3.11.1.4. Nitrogen OS

Es un proyecto basado en AOSP. Actualmente se concibe como un sustituto del software stock de teléfonos como el LG Optimus G, Nexus 4, Nexus 5, Nexus 6 y Nexus 6P.

Se trata de una ROM que siempre está actualizada a las últimas versiones de Android, compatible con el kernel personalizado ElementalX, y con variedad de configuraciones extra a las ofrecidas en una versión AOSP.

3.11.2. Distribuciones basadas en stock

Son distribuciones que se basan en la versión personalizada de Android desarrollada por cada fabricante, la cual en su mayoría es una capa de personalización. A pesar de que Android es un proyecto de código abierto, es probable que las capas de personalización y otro software indispensable que se encuentre en estas capas no sean libres, por lo que su modificación violaría los contratos a los que estuviera sujeto su uso.

Su función principal es "portar" versiones más actuales de la capa de personalización y de la versión de Android en las cuales se basan, permitiendo actualizar dispositivos más antiguos con software de terminales más nuevos.

Ejemplos de ellas son las ROM basadas en Touchwiz de Samsung, o en OxygenOS.

3.12. SafetyNet

Se trata de una implementación de APIs cuyo propósito es proteger a las aplicaciones desarrolladas contra amenazas. Ejemplos de estas amenazas son: modificaciones del software, aplicaciones maliciosas y URLs peligrosas. El rooting de un dispositivo es considerado una amenaza por SafetyNet, por lo que en principio hará que el dispositivo no supere la comprobación de seguridad. Sin embargo, desarrolladores de proyectos para obtener permisos "root" han tratado de implementar métodos para que se oculte a SafetyNet el estado modificado del teléfono.

3.13. ¿Por qué usar una ROM de la comunidad abierta?

Es importante enunciar las causas por las que un usuario optaría por usar una distribución desde la comunidad abierta. Las principales razones son:

→ Para tener un software más actualizado,

→ Para obtener nuevas características proveídas por una versión más nueva,

→ Obtener una experiencia del usuario de Android pura, sin software innecesario y sin capas de personalización,

→ Obtener más características que las existentes en las versiones stock del sistema operativo, y

→ Ser un usuario entusiasta que gusta de probar diferentes distribuciones.

Capítulo 4

Desarrollo

El dispositivo elegido, como fue mencionado con anterioridad, es el Motorola Nexus 6 XT1103 32GB P3A0. El soporte extendido para este terminal inició en enero de 2017, por lo que nuevas versiones de Android ya no serían distribuidas a éste por parte del fabricante. El soporte extendido por parte de Google para este móvil finalizó en octubre de 2017, con la versión 7.1.1 N6F27M que incluía el parche de seguridad del mismo mes tratado en el boletín de seguridad, aunque en abril se había lanzado ya una versión más nueva de Android (7.1.2) y en septiembre del mismo año, Android 8.0 fue publicado, por lo que las actualizaciones posteriores únicamente fueron relacionadas a la seguridad, y ya no proporcionaban nuevas características.

Las principales características del dispositivo son:

Chipset Qualcomm APQ8084 Snapdragon 805;

Procesador Quad-core 2.7 GHz Krait 450;

GPU Adreno 420;

Memoria 32/64GB, (no tiene SD card slot), RAM 3GB;

Bootloader con posibilidad de ser desbloqueado, no encriptado;

Pantalla 5.96 pulgadas AMOLED, touchscreen capacitivo, resolución 1440 x 2560 pixels, formato 16:9, densidad 493 ppi;

Batería 3220 mAh Li-Po;

Redes 4G, 3G y 2G;

Multimedia: Cámara trasera de 13MP con luz, y capacidad de grabación máxima de video en 4K@30Hz;

Conectividad Wi-Fi 802.11 a/b/g/n/ac, dual-band, Wi-Fi Direct, DLNA, hotspot, Bluetooth 4.1, A2DP, LE, GPS, con asistencia A-GPS, NFC. USB 2.0.

4.1. Instalación de una distribución Android de la comunidad abierta en el dispositivo

Para poder remplazar el software de fábrica de este dispositivo existen dos opciones: A través del software llamado Nexus Root Toolkit dentro de un entorno Windows, y por medio de comandos utilizando adb y fastboot.

Aunque el software mencionado anteriormente permite que el proceso se lleve a cabo dentro de interfaces gráficas, se mostrará cómo realizar todo el proceso usando comandos desde la terminal utilizando una disribución de Linux, aunque es posible utilizar Windows o Mac OS.

El proceso se compone de los siguientes pasos:

1. Seleccionar y descargar alguna distribución de Android desde la comunidad abierta,
2. Descargar las herramientas ADB,
3. Descargar alguna versión personalizada de Recovery desde la comunidad abierta,
4. Establecer una conexión del dispositivo con una PC por medio del cable USB 2.0,
5. Desbloquear el bootloader del dispositivo,
6. Modificar la partición Recovery mediante el remplazo de la versión original por el descargado en el paso 3,
7. Instalar la distribución de Android descargada en el paso 1,
8. Opcionalmente, instalar las aplicaciones relacionadas con Google,
9. Opcionalmente, obtener acceso de super usuario en la nueva distribución recién instalada.

Dentro del sitio `https://forum.xda-developers.com/nexus-6` encontramos las distribuciones disponibles para este dispositivo. Entre ellas, se encuentra la ROM Nitrogen OS.

4.1.1. Desbloqueo del bootloader e instalación de Recovery personalizado

Comandos adb y fastboot

Para cambiar el sistema operativo original en el dispositivo, es necesario desbloquear el bootloader, instalar un sustituto del Recovery para poder instalar el nuevo software por medio de un archivo .zip flasheable. Es posible obtener las herramientas que nos permiten comunicarnos con el dispositivo y enviar archivos a este, principalmente de tres maneras diferentes:

a) Descargar el IDE Android Studio desde `https://developer.android.com/studio/`. Este paquete incluye todas las herramientas así como el ambiente de programación oficial para desarrollar aplicaciones para Android, depurar, ejecutar emuladores, que es tratado en el capítulo 3. Las herramientas se encuentran, en el caso de Linux, en la ruta /home/Nombre_del_usuario/Android/Sdk/platform-tools que es creada al ejecutar por primera vez studio.sh que está dentro de la carpeta bin/ del paquete descargado.

b) Descargar únicamente el paquete SDK Tools desde el mismo sitio web, extraer el contenido del archivo en una ruta conveniente, y acceder a la carpeta "platform-tools".

c) Buscar dentro de los repositorios de nuestra distribución de Linux. En este caso, los paquetes necesarios son "android-tools", "android-sdk-platform-tools" y "android-udev". En una distribución basada en Arch Linux ejecutaríamos los comandos

```
$ sudo pacman -S android-tools
```

```
$ sudo pacman -S android-udev
```

```
$ sudo pacman -S android-sdk-platform-tools
```

para instalar las herramientas en el sistema.

Ahora bien, para que la comunicación con nuestro dispositivo usando el comando adb sea exitosa, es necesario:

→ Que el dispositivo esté encendido

→ Activar la Depuración por USB dentro de las Opciones de desarrollo, como lo muestra la figura 4.1.

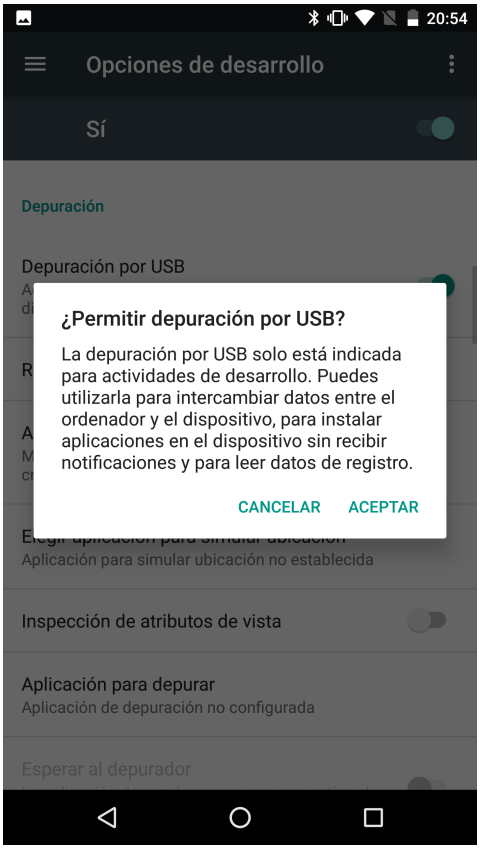

Figura 4.1: Activación de la depuración en el dispositivo.

Adicionalmente, activamos la opción Desbloqueo de OEM que nos permitirá desbloquear el bootloader, como lo muestra la figura 4.2.

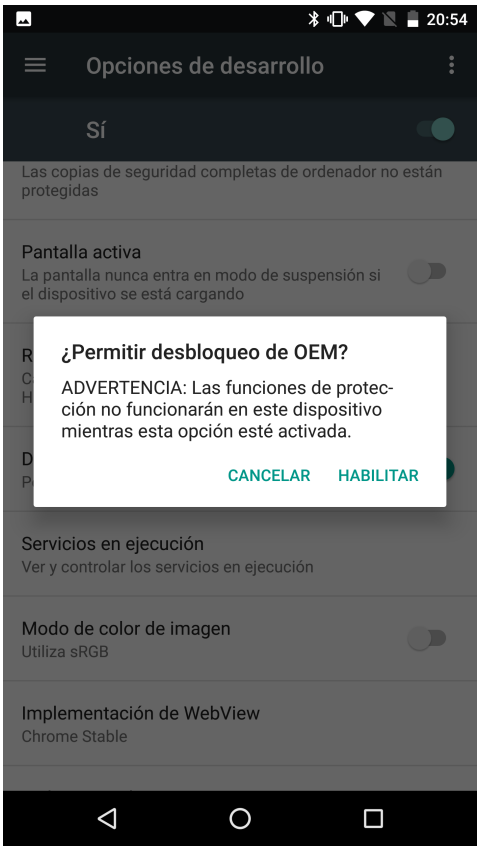

Figura 4.2: Permitir el desbloqueo OEM.

Una vez hecho esto, conectamos vía USB el dispositivo a la PC donde tenemos el SDK de Android. A continuación, debemos autorizar la PC en el dispositivo para poder usar la depuración vía USB.

Dentro de una terminal, nos desplazamos a la carpeta "platform-tools" y ejecutamos el comando

```
$ ./adb reboot bootloader
```

para reiniciar el dispositivo en modo Fastboot Flash Mode. Fastboot es el protocolo estándar de Android para instalar imágenes completas por medio de USB, pero no todos los dispositivos lo usan pues sus fabricantes han desarrollado sus propios protocolos. ODIN es el protocolo para móviles Samsung. Una vez que el dispositivo está en modo Fastboot, podemos leer en la pantalla del teléfono la leyenda "Device is locked" lo que

confirma que el bootloader actualmente está bloqueado, por lo que ejecutaremos el comando

```
$ sudo ./fastboot oem unlock
```

para desbloquear el bootloader. En la pantalla del dispositivo aparecerá un mensaje mostrado en la figura 4.3.

Presionamos el botón de encendido para continuar el desbloqueo, y en seguida la pantalla despliega una advertencia, mostrada en la figura 4.4.

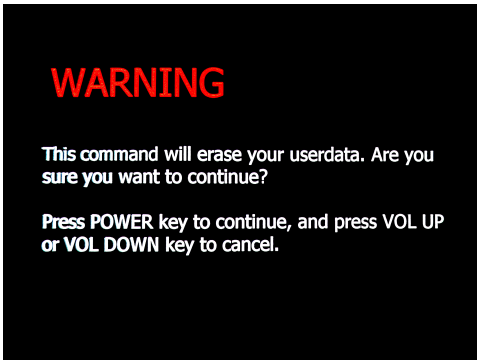

Figura 4.3: Desbloqueando el bootloader.

Presionamos el botón Volumen + y a continuación el botón de encendido. Después de esto, regresamos a la pantalla del modo Fastboot, y podemos verificar que es mostrada la leyenda "Device is UNLOCKED", por lo que el bootloader ha sido desbloqueado.

Recovery

Para continuar con el proceso de instalación de una distrbución, debemos modificar la partición /recovery para poder instalar la distribución desde el propio móvil, descargando previamente el archivo .zip que contiene la imagen del sistema.

Existe actualmente una opción de Custom Recovery (Recovery personalizado) para este modelo, desarrollado por la comunidad abierta, llamado TWRP (Team Win Recovery Project), que es un proyecto open source. Con anterioridad el proyecto Cyanogen-

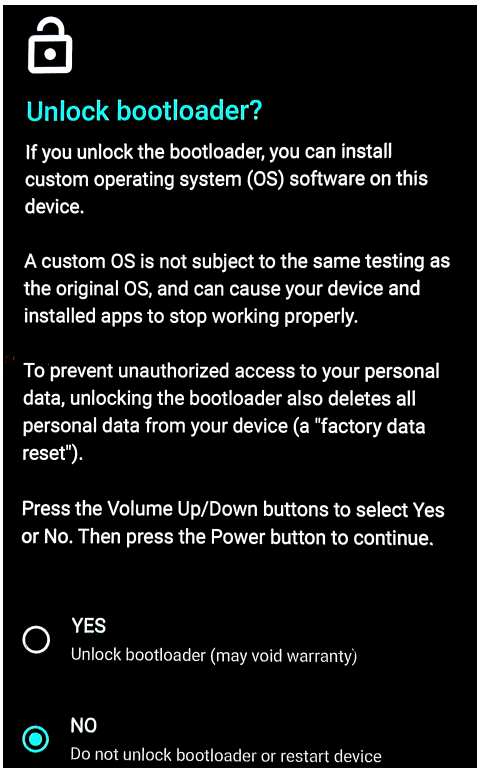

Figura 4.4: Advertencia al desbloquear el bootloader.

Mod desarrollaba su propio Custom Recovery, pero cuando el proyecto fue bifurcado, se continuó con el desarrollo de la distribución de Android por LineageOS, más no así con el Custom Recovery.

En la web de TWRP `https://twrp.me/` es posible buscar el software para el dispositivo. En este caso, el archivo "twrp-3.2.1-0-shamu.img" nos indica que la versión 3.2.1-0 es la versión más reciente para este móvil, por lo que descargamos el archivo y lo colocamos, de preferencia, en la misma carpeta donde están nuestras herramientas adb y fastboot. Para llevar a cabo el proceso de remplazo del Recovery, ejecutamos el comando con el dispositivo en modo Fastboot Flash Mode

```
$ sudo ./fastboot flash recovery twrp-3.2.1-0-shamu.img
```

cuya salida es mostrada en la figura 4.5.

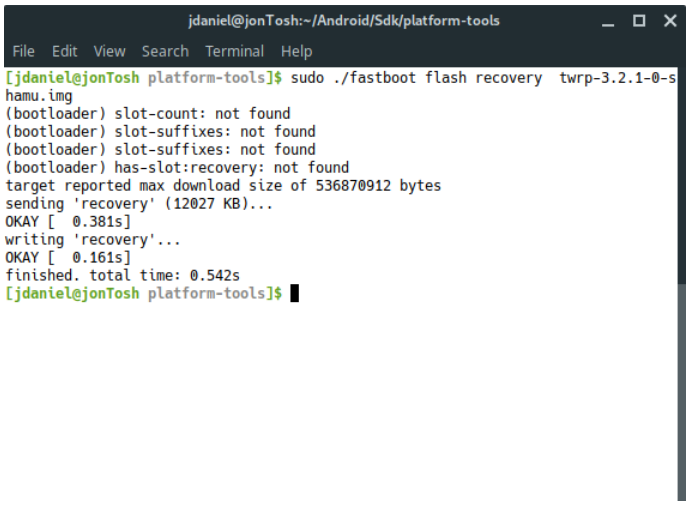

Figura 4.5: Salida del comando fastboot remplazando Recovery.

Inmediatamente después de enviar el comando para enviar el archivo del recovery, iniciamos el dispositivo en Modo Recovery siguiendo el siguiente procedimiento:

Procedimiento para entrar al modo Recovery: Presionamos el botón "-" del volúmen hasta llegar a "Recovery mode" y luego presionamos el botón de encendido.

A continuación el nuevo Recovery inicia, y debemos seleccionar si queremos que la partición /recovery se quede sin cambios o remplazar su contenido escribiendo de manera definitiva el recovery personalizado. En este caso se remplaza el contenido de la partición antes mencionada, deslizando el dedo hacia la derecha en la parte inferior de la pantalla.

Una vez hecho esto, el menú inicial del Recovery es mostrado, por defecto en idioma inglés (figura 4.6).

Descarga de una distribución de la comunidad abierta para el dispositivo

Ahora es necesario descargar la distrubución personalizada de Android. En este caso, fue seleccionada la versión Nitrogen OS por tratarse de un proyecto con un número significativo de opciones disponibles y compatibilidad con kernels persona-

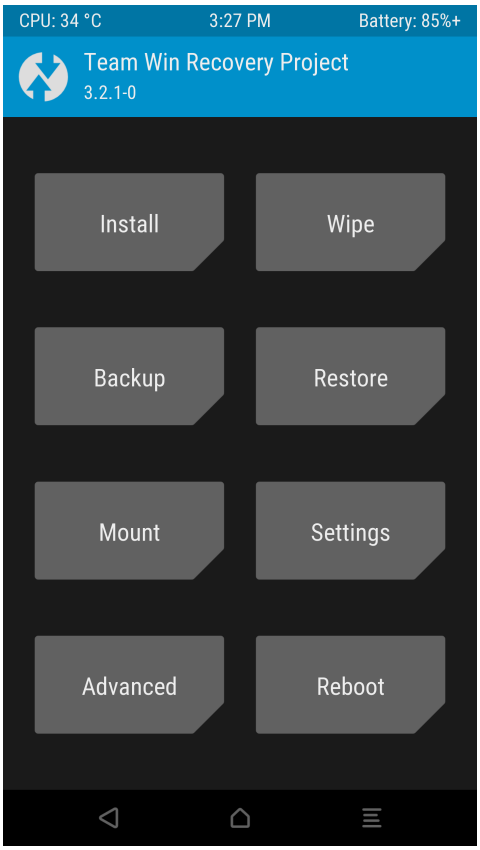

Figura 4.6: Pantalla principal del recovery TWRP.

lizados. En la página `https://forum.xda-developers.com/nexus-6/development/rom-nitrogen-os-n-substratum-11-10-2016-t3478365` están disponibles las descargas para el dispositivo Nexus 6, por lo que descargamos la versión que contiene el link interno, que corresponde a la última versión publicada por el equipo de desarrollo.

Al haber descargado el archivo de la distribución a instalar que está en formato zip, lo enviamos al almacenamiento del teléfono mediante una conexión USB. TWRP utiliza para la instalación archivos en formato .zip, por lo que no es necesario descomprimir su contenido.

4.1.2. Sustitución del sistema operativo

Para remplazar el sistema operativo, seguiremos los siguientes procesos:

Proceso 1: Borrar el sistema operativo actual - Debemos borrar el contenido de la partición /system que es donde se encuentra Android, borrar el contenido de caché de ART y borrar la partición /data, que es donde se encuentran instaladas las aplicaciones del usuario. Por lo tanto, debemos entrar al menú "wipe" y borrar las particiones mencionadas deslizando el dedo a la derecha en la parte inferior, donde dice "Swipe to Factory Reset" (figura 4.7). A continuación tocamos en el botón "Back" para regresar al menú principal.

Proceso 2: Instalar el .zip que contiene la nueva distribución de Android - Tocamos en "Install", seleccionamos la carpeta donde se encuentra el archivo .zip de la ROM, en este caso fue puesto en la carpeta /sdcard/Download/. Una vez ahí, tocamos en el archivo zip y confirmamos haciendo swipe con el dedo en donde dice "Swipe to confirm Flash" (figura 4.8). En la pantalla, podemos ver un resumen de los procesos llevados a cabo de acuerdo a un script dentro del archivo .zip (figura 4.9), por lo que es necesario esperar a que el proceso de instalación termine. Una vez finalizado, podemos tocar en "Reboot System" y el móvil será reiniciado.

Veremos la animación que aparece al iniciar el nuevo sistema, y al finalizar, vemos la pantalla de inicio del lanzador de aplicaciones. Hasta este momento, tenemos ya una distribución diferente en nuestro teléfono a la que tenía de fábrica. Para corroborar qué versión de Android tenemos, vamos a la configuración y entramos en Sistema, y a continuación tocamos "Información del teléfono". La información de la figura 4.10 contiene la versión de Android actual, la versión de la ROM y el nivel del parche de seguridad de Android, que en este caso es de julio de 2018.

Aunque tenemos una versión funcional de Android, no se encuentra instalado ningún servicio de Google, como la tienda de aplicaciones Play Store, ni el cliente de correo de Gmail. Por ahora, sólo están instaladas las aplicaciones básicas que permiten el funcionamiento básico del teléfono (figura 4.11):

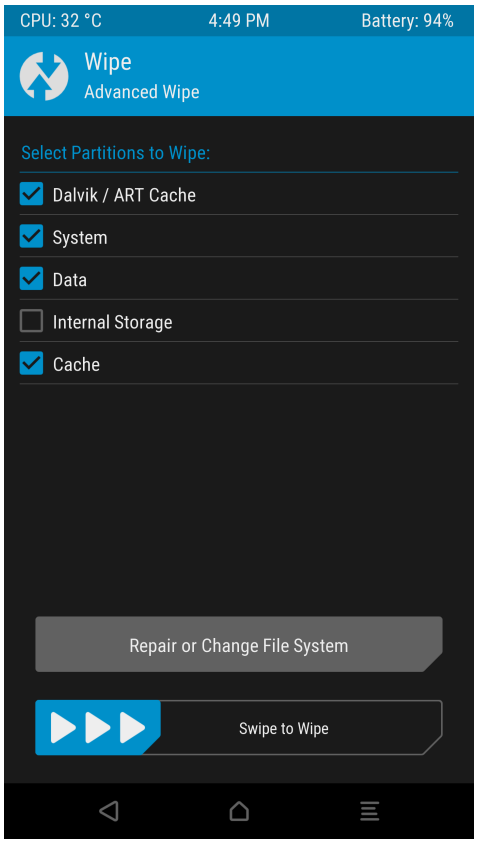

Figura 4.7: Menú avanzado del formateo de particiones.

→ Aplicación de teléfono y contactos

→ Aplicación de SMS

→ Aplicación de cámara

→ Calendario

→ Calculadora

→ Galería

→ Reloj

Para instalar nuevas aplicaciones es necesario instalar una tienda de aplicaciones. Play Store es la tienda de aplicaciones de Google, y es la que posée el mayor número

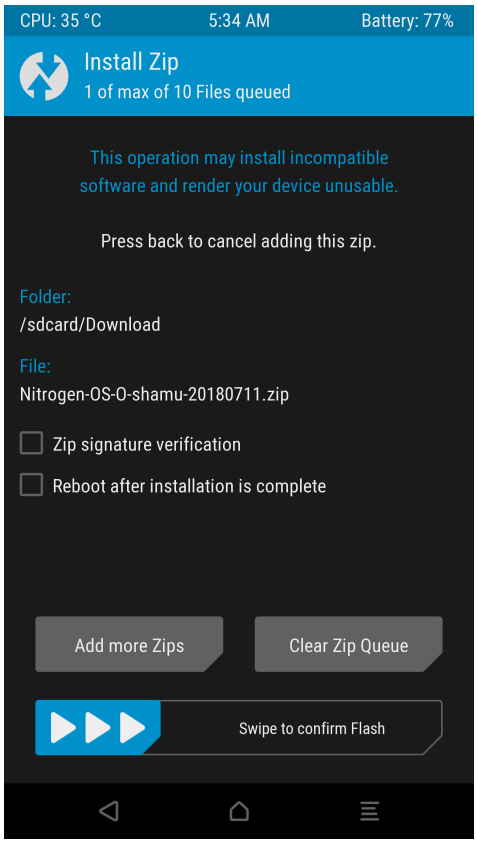

Figura 4.8: Confirmación de la instalación haciendo Swipe.

de aplicaciones para dispositivos Android. Para obtener Play Store, es necesario que nuestro dispositivo tenga los servicios de Google instalados. Por problemas de licencias de uso, ninguna ROM basada en AOSP puede contener algún servicio de Google. Las "Google apps" son aplicaciones propietarias de Google, por lo que no son de código abierto, y es por ello que la distribución que fue instalada no las incluye, aunque es posible descargarlas e instalarlas sobre nuestro nuevo sistema.

4.1.3. OpenGapps

Es una recopilación actualizada no oficial disponible públicamente de las aplicaciones de Google para Android. Son paquetes que incluyen desde los servicios básicos para poder acceder a la tienda de aplicaciones, hasta un repertorio completo de todas y cada

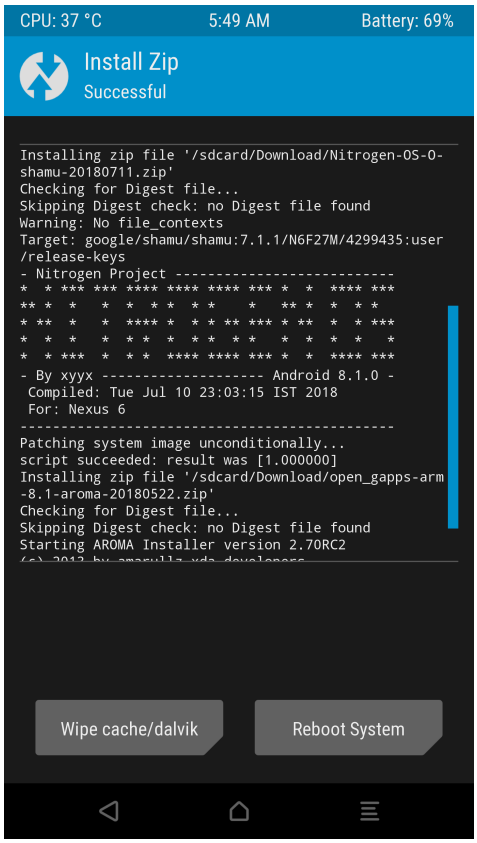

Figura 4.9: Resumen de instalación de Nitrogen OS

una de las aplicaciones de Google que se instalan por medio de un script automatizado, o un asistente llamado "Aroma" que permite la selección de las aplicaciones, servicios y tareas post-instalación.

Como existen plataformas ARM, ARM64, x86 y x86-64, están disponibles para cada una de ellas, y hay para cada una de ellas, paquetes con un número diferente de aplicaciones. Se encuentran clasificados en:

tvstock Contiene las Google apps incluídas en el último Nexus Player

pico El mínimo de aplicaciones para obtener funcionalidad de Google Play

nano El mínimo más servicios no disponibles en Play Store

micro Lo mismo que nano más Gmail, Calendar y Google Now

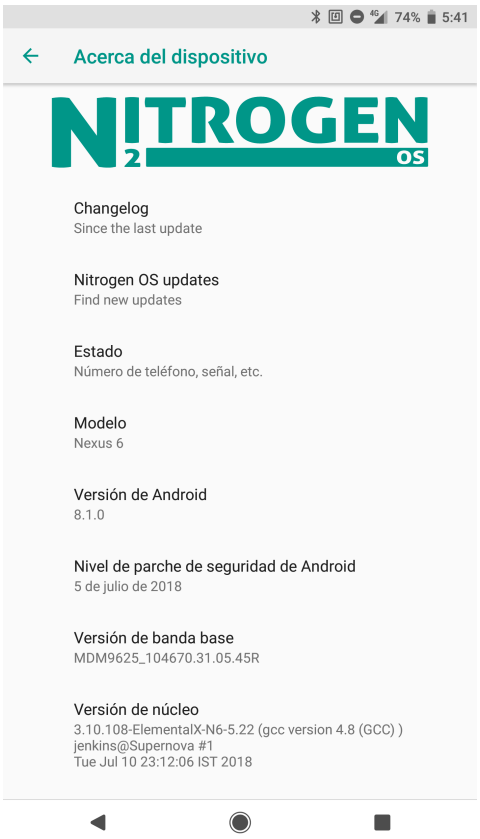

Figura 4.10: Información acerca del sistema operativo.

mini Igual que micro, más otro set de aplicaciones de Google

stock Todas las aplicaciones contenidas en los dispositivos más recientes Nexus y Pixel

full Incluye las aplicaciones de stock, pero no remplaza las nativas que no son propietarias de Google

super Todas las Google Apps

aroma Incluye el instalador Aroma que permite personalizar las aplicaciones instaladas y remplazadas, además de crear copias de seguridad.

En la página `https://opengapps.org/#aboutsection` se informa que cualquier descarga no incluye licencia alguna para el uso de las aplicaciones. De cualquier manera,

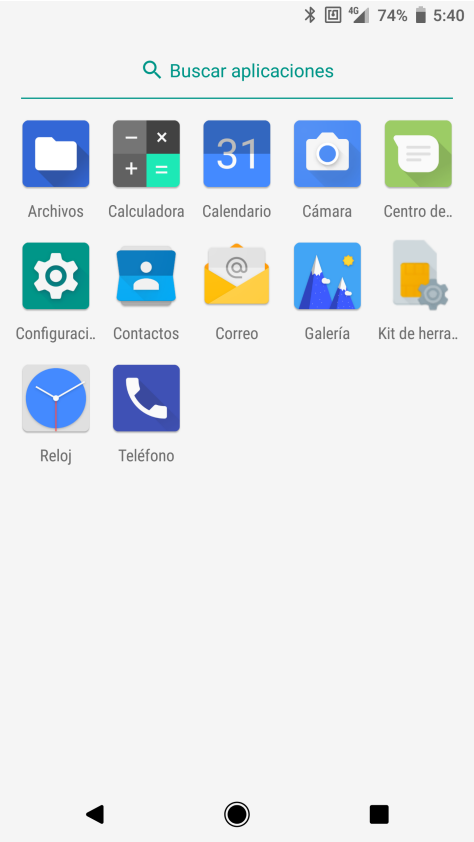

Figura 4.11: Aplicaciones básicas incluídas con la ROM

el dispositivo usado en esta tesis traía preinstalados los servicios de Google, por lo que es un dispositivo con licencia.

Para instalar las aplicaciones en el Nexus 6 dentro de la nueva distribución Android, dentro de un navegador vamos a la página `https://opengapps.org/`. Seleccionamos el paquete ARM para la versión 8.1 y por último, seleccionamos la variante con el instalador Aroma. La descarga es de 1030MB, como lo muestra la figura 4.12.

El proceso de instalación de las aplicaciones de Google es el mismo que el "Proceso 1" que seguimos anteriormente para instalar Nitrogen OS, por lo que una vez que hayamos descargado el .zip que correspondiente a la versión seleccionada, lo enviamos al móvil, y reiniciaremos el dispositivo en modo "Recovery" presionando simultáneamente los botones de "bajar volumen" y "encendido/apagado" por diez segundos. Enseguida

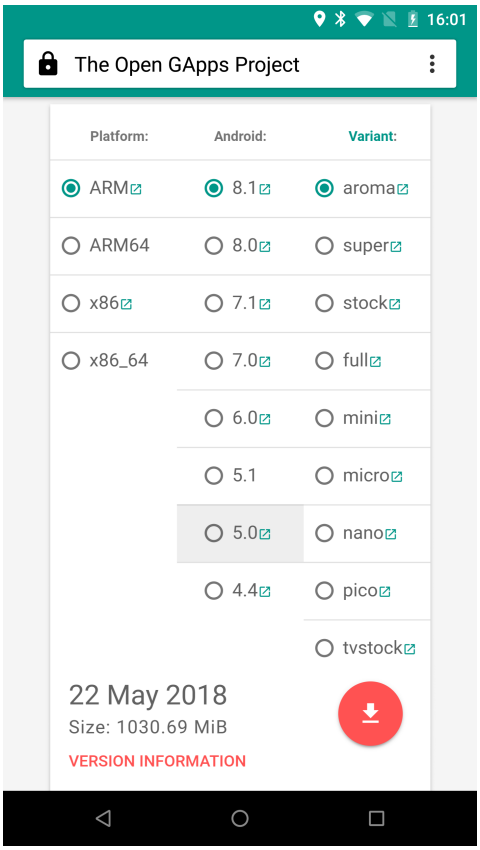

Figura 4.12: Descargas disponibles de Google Apps

estaremos en modo Fastboot, por lo que seguimos el mismo procedimiento seguido con anterioridad para entrar al modo Recovery desde el modo Fastboot. Una vez dentro del modo Recovery, seguimos el mismo procedimiento para flashear un .zip, pero seleccionamos el archivo que contiene las Google Apps. En este caso, se inicia el instalador AROMA, como lo muestra la figura 4.13.

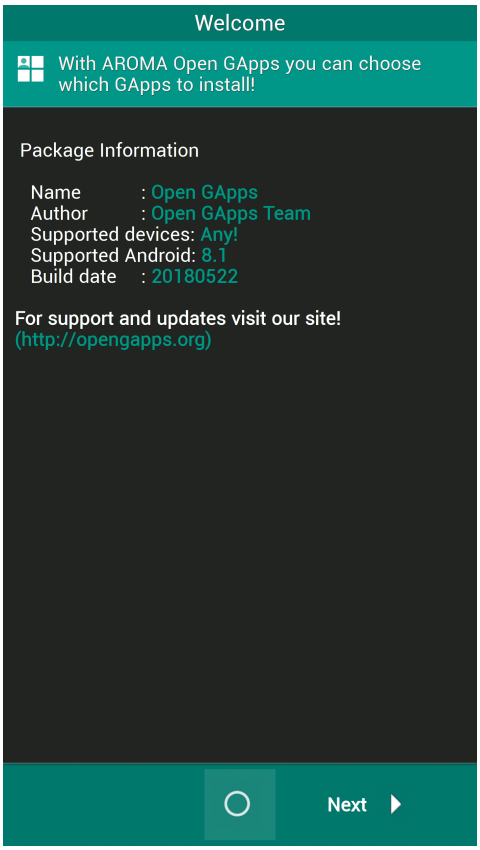

Figura 4.13: Instalador de las Google Apps.

El instalador AROMA es un asistente que permite la personalización de los componentes que se instalarán. Este asistente es usado frecuentemente en la comunidad con otras instalaciones.

El proceso consiste de los siguientes pasos:

1. Seleccionamos "Customized installation" para seleccionar qué aplicaciones y configuraciones serán aplicadas.

2. La opción "Load choices" no funcionará porque no se han instalado antes estas aplicaciones en el móvil, entonces no hay un archivo previo.

3. En la siguiente pantalla, elegimos la opción "Exclude" para seleccionar las aplicaciones que no queremos que sean instaladas. Las que no sean seleccionadas, serán

instaladas. En este caso, instalaremos Calendar, Calendar Sync, Camera, Carrier Services, Chrome, Clock, Contacts, Dialer framework, Google Dialer, Exchange Services, Google Conectivity Services, Gmail, Text-to-Speech, Keep, Keyboard, Maps, Messages, Pixel Icons*, Pixel Launcher*, Photos, Search, Translate, VR Service y Wallpapers.

4. En la siguiente página del asistente, seleccionamos qué apps AOSP serán ignoradas para ser remplazadas por las de Google, por ejemplo el navegador web.

5. El penúltimo paso del asistente nos permite remover otros componentes, y en el último paso podemos activar opciones avanzadas:

→ Creación de una bitácora

→ Simular la instalación generando un registro completo, sin hacer cambios

→ Omitir la instalación y remplazo de las librerías para el teclado AOSP.

→ Deshabilitar la generación inteligente Pre-ODEX.

→ Forzar la instalación

→ Agregar al archivo build.prop el soporte para Google Assistant.

A continuación, confirmaremos los cambios y la instalación iniciará. Una vez finalizada, salimos de AROMA y después reiniciamos el equipo. El proceso de inicio tardará algunos minutos, ya que los archivos caché de ART serán creados para las nuevas aplicaciones. Al haber finalizado el inicio del sistema, se nos preguntará sobre cuál lanzador de aplicaciones queremos utilizar. El lanzador instalado en el asistente fue Pixel Launcher, por lo que seleccionamos ese como predeterminado.

4.2. Configuraciones adicionales

4.2.1. Root en la nueva distribución

A pesar de que la distribución instalada en el dispositivo permite la configuración de más parámetros que el software oficial, ganar acceso de súper usuario supone un aumento de las posibilidades de personalización por medio de software de terceros.

Tener acceso root en el dispositivo permite que los parámetros del kernel incluído sean modificados, permitiendo que un usuario con conocimientos suficientes cambie valores que en otro caso no podrían ser personalizados. También es posible remplazar el kernel por alguno con características diferentes.

Se utilizará un método automatizado de rooting, el cual requiere que el bootloader esté desbloqueado y que exista instalada una versión personalizada de Recovery.

Las dos opciones más comunes son SuperSU y Magisk, que proveen un root Systemless.

Estas herrmientas han permitido a las aplicaciones que necesitan permisos de súper usuario, obtengan los privilegios con la previa autorización del usuario, similar a lo que sucede en alguna distribución Linux al tratar de ejecutar un comando con permisos root.

4.2.1.1. SuperSU

Es un proyecto desarrollado inicialmente por Chainfire, pero que hace tiempo comenzó un proceso de traspaso a una empresa cuyos orígenes no están muy claros, por lo que podría ser sujeto de cuestionamientos, dado que el propio propósito de la aplicación, aunque en la página `http://www.supersu.com/about#privacy-policy-supersu` se encuentra su declaración de privacidad, en idioma inglés.

Se caracteriza por ser compatible con los Custom Recovey como TWRP y Clockworkmod, proveer permisos individuales para cada proceso y cada aplicación que requiera acceso de privilegios, guardar en un log qué aplicaciones y qué procesos hacen uso de permisos elevados, recordar permisos para una aplicación, y revocarlos si es necesario. Provée de actualizaciones frecuentes, y es compatible con casi todas las distribuciones de Android personalizadas, así como las stock.

4.2.1.2. Magisk

Magisk es un proyecto OpenSource más reciente que Supersu y que es compatible con una gran cantidad de dispositivos. Su código está disponible en Github en la página `https://github.com/topjohnwu/Magisk`. Permite la instalación de módulos que modifican o añaden funciones al sistema operativo.

El software utilizado para ganar permisos root será Magisk, el cual proporciona un root systemless tratado en el capítulo anterior. Para ello, iremos al sitio `https://forum.xda-developers.com/apps/magisk/official-magisk-v7-universal-systemless-t3473445` que es un hilo donde hay un .zip instalable por medio de Recovery. En la sección titulada "Downloads", hacemos clic en "Latest Magisk". Al finalizar la descarga, reiniciamos al modo Recovery, e instalamos el archivo instalable .zip. La salida del script es mostrada en la figura 4.14.

Después de reiniciar el sistema, estará disponible un administrador de permisos root, por medio de la aplicación Magisk Manager.

4.2.2. Utilización de un kernel personalizado

El kernel incluído en esta distribución fue personalizado y no es el original incluído en la versión stock del software por parte del fabricante, aunque proviene de la misma rama que el fabricante desarrolló para el dispositivo, en favor de mantener compatibilidad con

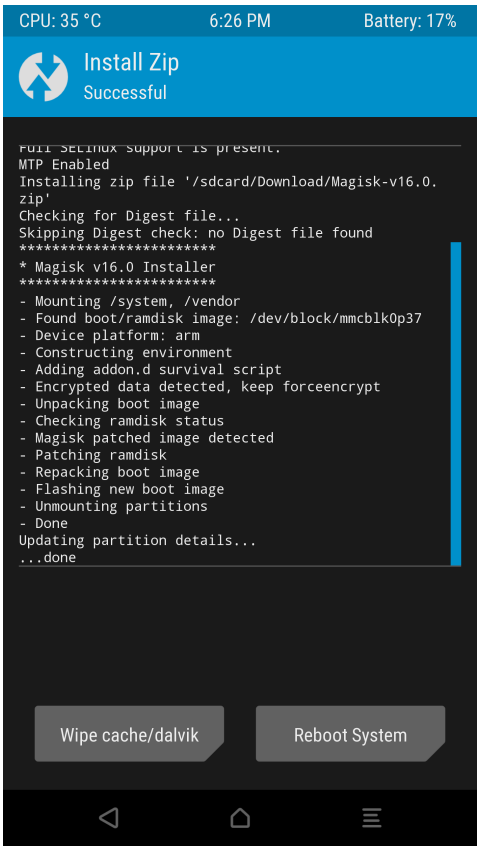

Figura 4.14: Salida del script de instalación de Magisk.

los drivers.

Diferentes proyectos de personalización de kernel han existido desde la aparición del HTC Dream, los cuales proporcionan al usuario la capacidad de personalizar configuraciones específicas del kernel, de manera similar a los sistemas de escritorio Linux, pero que no se pueden modificar en un sistema con el software original.

Los proyectos de personalización de kernels más importantes para el Nexus 6 han sido:

→ Franco Kernel

→ ElementalX Kernel

→ LeanKernel (descontinuado)

Éstos han sido reconocidos por la comunidad por la continuidad de desarrollo, su estabilidad, y no menos importante, las nuevas características que cada uno de ellos proporciona al usuario. Entre estas características se encuentran:

→ Gobernador del CPU configurable

→ Gobernador del GPU

→ Over-clocking y under-clocking

→ Activación y desactivación de núcleos

→ Cambio de voltajes del procesador

→ Personalización de los planificadores de E/S

→ Configuración de "Low memory killer"

→ Manejo de gráficos y gestos en pantalla

A pesar de ofrecer actualizaciones periódicas relacionadas a las características añadidas, cualquier kernel personalizado no ofrece actualizaciones respecto a la versión del kernel Linux del cual derivó. El objetivo de estos proyectos es permitir que el usuario obtenga más posibilidades de personalización, y no participan en el desarrollo directo de la rama del kernel de Android. El kernel del cual derivan todas las versiones para este móvil es la versión 3.10.x. La última actualización oficial del kernel 3.10, cuya versión es la 108 se lanzó el 4 de noviembre de 2017 y es la que incluye el kernel utilizado en esta distribución.

El kernel que incorpora esta distribución proviene del desarrollo del kernel ElementalX, en su versión 5.22 por lo que no es necesario realizar el proceso de instalación nuevamente. Para obtener una versión actualizada de este kernel personalizado y cambiar sus características, es necesario comprar la aplicación EX Kernel Manager de la

tienda de aplicaciones Play Store, cuyo costo es de treinta pesos mexicanos aproximadamente.

4.2.2.1. Kernel manager

EX kernel manager es una aplicación que permite la modificación de parámetros configurables del kernel. En la tienda Play Store hay aplicaciones similares como Kernel Auditor o Kernel Tuner, las cuales pueden monitorear y crear un log del uso del procesador por las aplicaciones, permiten la creación de perfiles y visualización del performance actual del equipo, como memoria usada, carga del CPU, frecuencias usadas, etc.

4.2.3. Configuraciones disponibles del Kernel

4.2.3.1. CPU

Gobernador del CPU: Se trata de un plan en el cual se determina el estado y la frecuencia de los núcleos del procesador de acuerdo a entradas como la carga actual de trabajo por núcleo y la frecuencia actual. En Android, el más común ha sido "interactive" aunque están disponibles más gobernadores (figura 4.15). Los gobernadores tienen parámetros que determinan cuándo un núcleo es activado y su frecuencia es aumentada. Estos valores son expuestos al usuario con permisos de root en Android, según lo muestra la figura 4.16.

El kernel proporciona un gobernador adicional llamado "elementalx" desarrollado por el autor de este kernel, y este permite la definición de una frecuencia máxima de CPU cuando la pantalla está apagada, un up_threshold (porcentaje de carga del procesador máxima antes de aumentar la frecuencia) predeterminado de 90, y otros valores definidos por el desarrollador (ver figura 4.17).

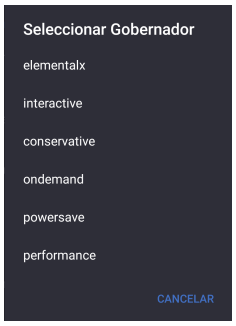

Figura 4.15: Lista de gobernadores disponibles en el kernel ElementalX

Otras opciones configurables relacionadas al manejo del CPU son el manejo térmico. Por defecto, el CPU es ralentizado cuando se alcanza una temperatura elevada por medio del driver msm_thermal, core_control y vdd_restriction (figura 4.18).

También es posible la modificación de parámetros de "hotplugging" del CPU (activación/desactivación de cores). Es posible crear un plan personalizado estableciendo cualquiera de las propiedades mostradas en la figura 4.19. Aquí podríamos establecer por ejemplo, que el número de cores máximos activos cuando la pantalla está apagada sea 2, o que el número de cores máximos encendidos sea otro número diferente al predeterminado.

4.2.3.2. GPU y Pantalla

Únicamente es posible cambiar las frecuencias base y máxima del GPU, y elegir un gobernador diferente de la lista, en donde fueron añadidos otros más por parte de la comunidad abierta, entre los cuales encontramos "msm-adreno-tz" como predeterminado, pero también están "userspace", "powersave", "performance", "simple_ondemand", "msm_cpufreq".

Adicional a estas configuraciones, los colores mostrados en pantalla pueden ser personalizados variando la intensidad de rojo, verde, azul, saturación, contraste y matiz en valores de 8 bits (0-255). Además es posible cargar perfiles ya contenidos en el kernel,

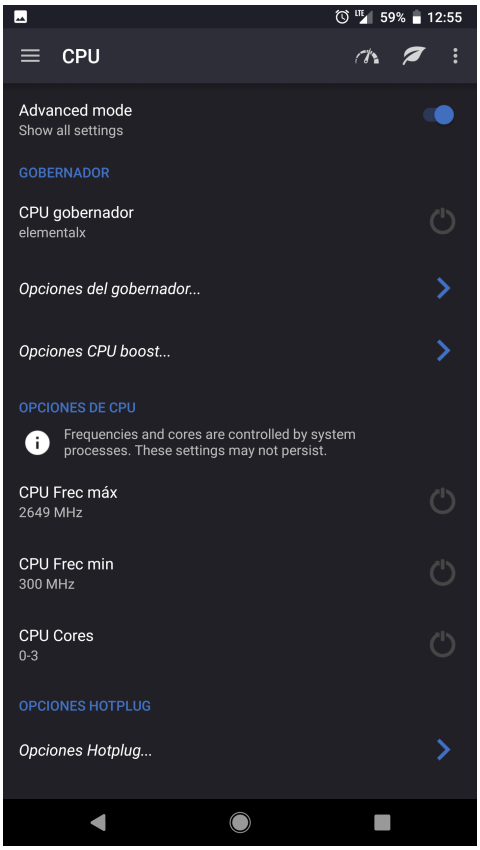

Figura 4.16: Opciones relacionadas al apartado CPU

o crear nuevos.

En la sección llamada "Gestos" son añadidos comportamientos cuando tocamos la pantalla o realizamos algún gesto. Por ejemplo, cuando la pantalla está apagada, es posible ir a la pantalla de desbloqueo sin tocar algún botón tocando dos veces la pantalla. También es posible configurar el bloqueo por medio de swype, el lanzamiento de la app de la cámara con un gesto, y configurar una retroalimentación después de un gesto mediante una vibración de intensidad configurable (figura 4.20).

4.2.3.3. Voltaje

En esta sección se podría modificar el valor de la energía entregada en miliVolts al CPU de acuerdo a la frecuencia de funcionamiento de cada núcleo. Se trata de una ca-

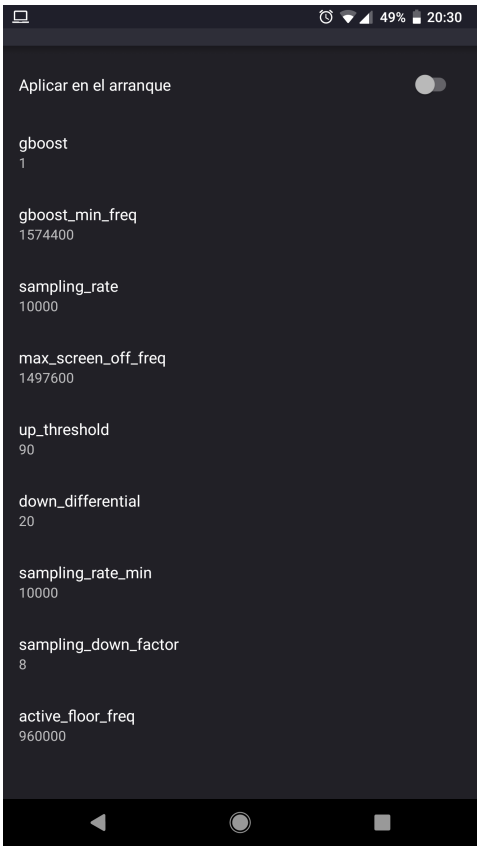

Figura 4.17: Opciones configurables del gobernador elementalx.

racterística común en equipos de escritorio frecuentemente al practicar "overclocking". Al modificar estos parámetros se corre el riesgo de dañar el hardware u obtener un comportamiento inestable del equipo, pero los usuarios experimentados podrían explotar a su propia consideración.

4.2.3.4. Audio

Sólamente podremos configurar alguna ganancia a las salidas de los audífonos, y a las entradas de la videocámara y el micrófono en llamadas.

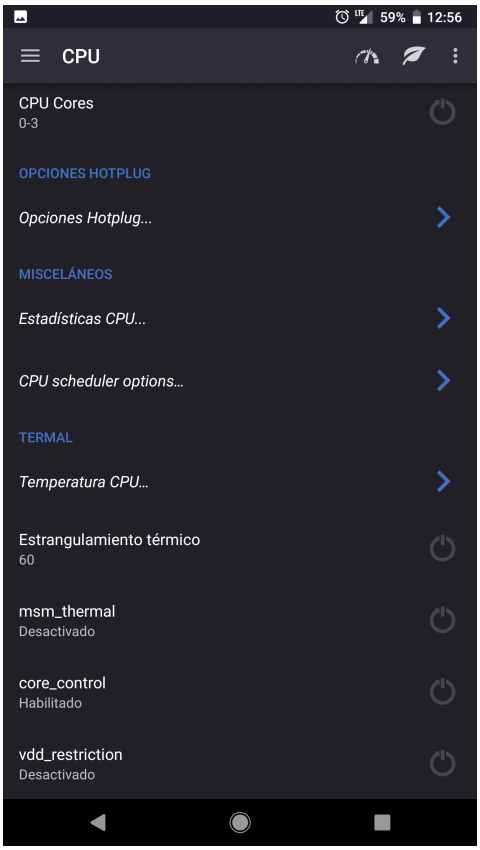

Figura 4.18: Otras opciones configurables del gobernador elementalx.

4.2.3.5. Memoria

Aquí se encuentran dos subsecciones relacionadas al uso de memoria RAM: "Low memory killer" y las opciones de la memoria virtual.

Low Memory Killer (LMK) es un proceso que tiene como objetivo terminar la ejecución procesos de acuerdo a su prioridad en el sistema, mediante el valor de kernel *oom_score_adj*, que define un rango de puntuación desde -17 hasta 15, siendo el máximo valor asignado a los procesos con mayor probabilidad de ser terminados. Encontramos seis categorías de procesos:

1. Aplicaciones en primer plano: Es la aplicación que está en ejecución y es mostrada en la pantalla.

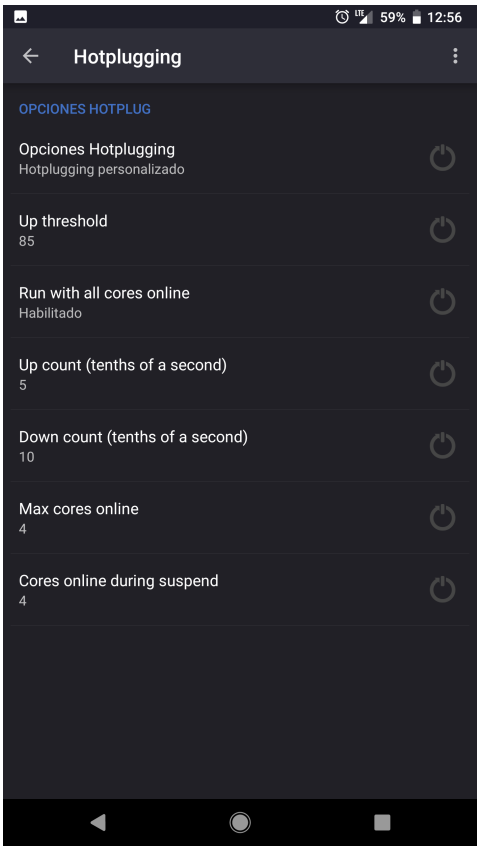

Figura 4.19: Hotpluging personalizado.

2. Aplicaciones visibles: Son aplicaciones que pueden no ser mostradas en la pantalla actualmente pero siguen en ejecución. Pueden tener ventanas transparentes.

3. Servicios secundarios: Son los servicios que se ejecutan en "background" y son necesarios para que ciertas aplicaciones se ejecuten correctamente. Un ejemplo es "Google Play Services".

4. Aplicaciones ocultas: Son aplicaciones no mostradas al usuario pero corren en background.

5. Content providers: Son servicios que proveen contenido al sistema, como ubicación, contactos, etc.

6. Aplicaciones vacías: Son aplicaciones que el usuario abandonó pero son manteni-

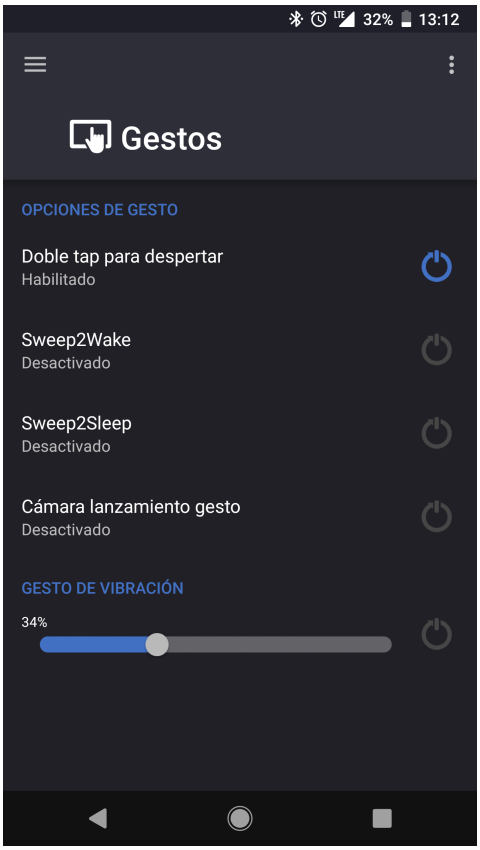

Figura 4.20: Gestos incluidos en el kernel

das en memoria, aunque no están en ejecución, por lo que no ocupan tiempo de procesador.

Dicho esto, a las aplicaciones vacías se les asigna un valor *oom_score_adj* elevado, mientras que a las aplicaciones en primer plano se les asignaría un valor bajo.

En la sección "Low Memory Killer" podremos modificar el valor que representa la cantidad de memoria libre a la que, dependiendo de la categoría de procesos, LMK comenzará a terminar procesos (ver figura 4.21).

Android Activity Manager se encarga de asignar el valor *oom_score_adj* a cada categoría en el proceso de arranque del sistema. Hay procesos críticos que no pueden ser terminados por LMK.

Sobre las opciones de la memoria virtual, es posible configurar:

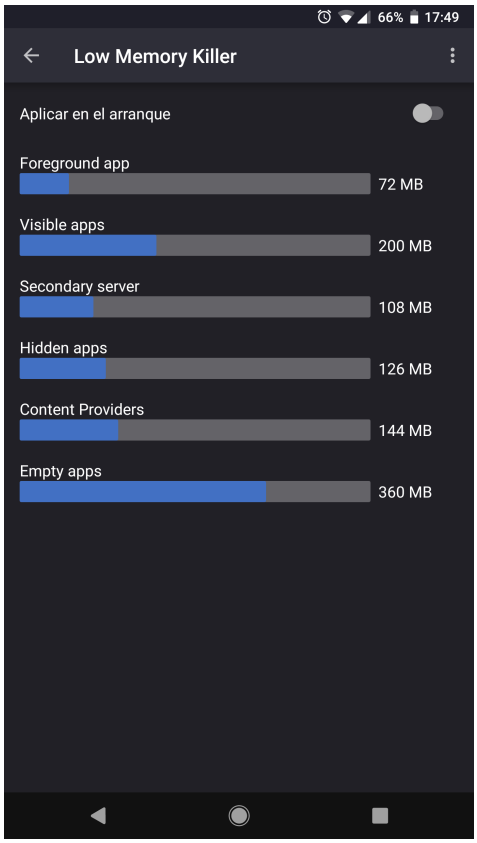

Figura 4.21: Configuración del LMK.

1. "dirty ratio": Representa al porcentaje de memoria total del sistema, que cuando es superado, es forzada la escritura de datos "de caché" al medio de almacenamiento no volátil por medio de pdflush (demonio que se encarga de vaciar la caché escribiendo su contenido en disco).

2. "dirty background ratio": También representa un porcentaje del valor de memoria total, y define el monto de memoria cuando comenzará la escritura de datos de caché al medio de almacenamiento no volátil en segundo plano vía pdflush.

3. "min free kbytes": El valor mínimo de memoria en kilobytes que debe mantenerse libre. Permite que el sistema mantenga una cantidad adecuada de memoria caché en caso de requerirla inmediatamente, sin esperar en caso de que algún proceso

así lo requiera.

4. “vfs cache pressure”: Es un valor que controla la tendencia del kernel a solicitar que la memoria usada en caché VFS (caché del sistema de archivos) sea liberada. El valor predeterminado es 100.

5. “overcommit ratio”: Para explicar este término es necesario hacer notar que *overcommit_memory* es un parámetro del kernel que puede tomar tres valores (0, 1 y 2), y cuya función es permitir la sobrecarga de memoria ocasionada por los requerimientos de las aplicaciones ejecutadas. Si esta opción es 0, el kernel se encargará del manejo de la sobrecarga de memoria, pero si es 1 el kernel permitirá la sobrecarga pero no se hará cargo de ella. En el caso que este valor sea 2, el kernel negaría peticiones iguales o mayores a la suma de la memoria swap disponible más el porcentaje de la memoria RAM (física) disponible especificado en overcommit_ratio. Esta opción a pesar de ser mostrada en las opciones del kernel, requiere que el parámetro overcommit_memory=2. Para consultar qué valor tiene “overcommit_memory”, en una ventana de terminal ejecutamos el comando

   ```
   cat /proc/sys/vm/overcommit_memory
   ```

   El resultado es mostrado en la figura 4.22.

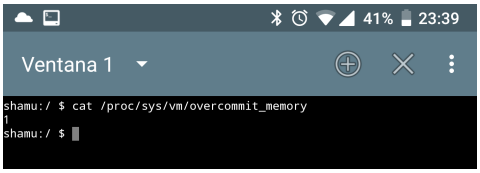

Figura 4.22: Valor de overcommit_memory en el Nexus 6.

Es posible observar que el valor de “overcommit_memory” es 1, por lo que “overcommit_ratio” no es útil. Adicionalmente, cambiarlo a 2 no tendría algún efecto porque Android no usa una memoria de intercambio “swap”.

4.2.4. Configuraciones adicionales del kernel

Dentro de la app EX Kernel Manager encontramos una sección con otras configuraciones:

→ Planificador I/O: Es el componente del kernel que optimiza la entrega de operaciones de entrada/salida a los dispositivos de almacenamiento. Este kernel ocupa por defecto al planificador NOOP, el cual introduce las peticiones de entrada/salida en una cola FIFO (First In, First Out) el cual es considerado ser el algoritmo más sencillo. El kernel original de este dispositivo usaba el algoritmo CFQ el cual se caracteriza por compartir por igual el ancho de banda a las operaciones de entrada/salida. Otros planificadores son: deadline, row, bfq y zen, etc.

→ "readhead kb": es el valor en kilobytes de datos que el kernel lee de forma extra del dispositivo de almacenamiento o de la memoria caché. Es un mecanismo de predicción que se usa para disminuir las operaciones de entrada/salida. Valores entre 128 hasta 4096 kilobytes pueden ser seleccionados.

→ "fsync": Se encarga de copiar las partes de un archivo que está en caché al almacenamiento. Por defecto se encuentra activado.

→ Vibración: Ajusta un valor de intensidad de vibración de los eventos que la usan, por ejemplo, la retroalimentación de algún gesto.

→ Carga rápida USB: Permite que la intensidad de corriente sea aumentada cuando se usa un puerto USB de la computadora para cargar la batería.

→ Algoritmo de congestión TCP: La congestión TCP ocurre cuando el emisor envía paquetes a alta velocidad sin conocer la capacidad de recepción del receptor.

Esto resulta en pérdida de información y disminuye el rendimiento de transferencia. Por ello, los algoritmos de congestión proveen de estrategias dependiendo de las condiciones de carga de la red. La latencia puede variar de acuerdo al algoritmo seleccionado. En este kernel están disponibles: cubic (predeterminado), westwood, reno, vegas, veno, illinois, etcétera. Westwood es un algoritmo que ha sido recomendado por la comunidad abierta porque ofrece una baja latencia.

→ wakelocks: Es una característica que permite a los desarrolladores que el dispositivo entre en modo de reposo para que así una tarea sea completada. Aunque existen alternativas para evitar que los desarrolladores usen indiscriminadamente esta característica, es posible elegir y bloquear alguno en la lista. En la lista hay dos wakelocks activados: msm_serial_hs_dma-144 y smb_135x_wake.

→ Entropía disponible: representa el valor disponible de los 4096 bytes de valor "entropy pool size" . Los valores *read_wakeup_treshold* y *write_wakeup_threshold* pueden ser modificados. Se tratan de valores que pueden ser modificados si necesitamos cambiar nuestras demandas de entropía en el sistema.

4.3. Pruebas de la nueva distribución

Como se pretende analizar el comportamiento del dispositivo utilizado en este trabajo después de instalar una distribución de Android desde la comunidad abierta, el siguiente paso consiste en analizar el comportamiento del sistema evaluando la estabilidad proporcionada al usuario, para comprobar que puede ser una alternativa para mantener el teléfono en un estado utilizable.

4.3.1. Aplicaciones a instalar

Después de haber instalado las aplicaciones de Google (GAPPS), descargamos desde Play Store: Adobe Acrobat, Instagram, Magisk Manager, Facebook, Facebook Messenger, Mobile Print Samsung, Nissan DataScan II, Opera, Shazam, Spotify, Traductor, VLC, Waze, WhatsApp y Youtube.

Como esta distribución es basada en AOSP, las aplicaciones debieron haber sido probadas en emuladores y no deberían tener problemas en ser ejecutadas en el terminal. Es de destacar que de todas las aplicaciones instaladas desde la tienda de aplicaciones, únicamente WhatsApp advertía que, a causa de estar utilizando una ROM personalizada, no obtendríamos soporte en caso de problemas con esta app (ver figura 4.23).

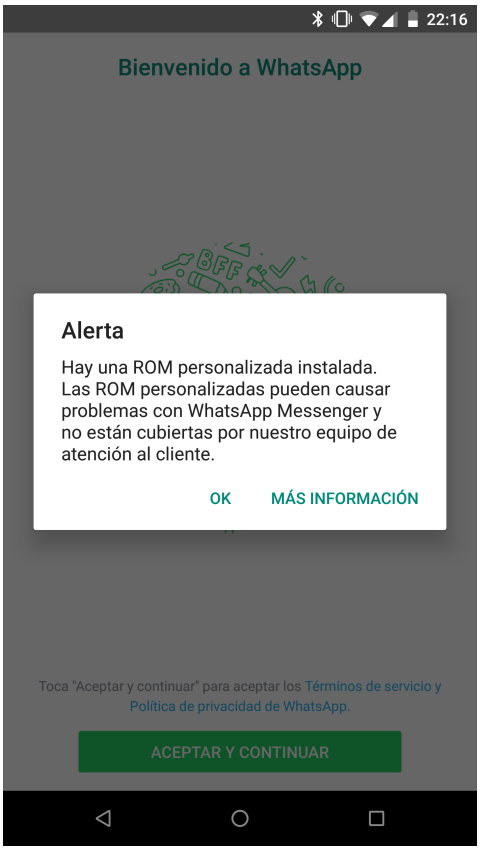

Figura 4.23: Whatsapp no ofrece soporte en ROM personalizadas.

4.3.2. Benchmarks

Un benchmark se trata de la medición del rendimiento del dispositivo, principalmente, del CPU, GPU, memoria y medio de almacenamiento.

Usaremos la aplicación Antutu Benchmark, que es gratuita y se puede obtener desde la tienda de aplicaciones Play Store, la cual es ampliamente utilizada dentro de la comunidad abierta. Esta app genera un score al finalizar las pruebas de rendimiento. Dicho score representa la suma de varias pruebas y pueden ser obtenidos los resultados de cada prueba individualmente con el fin de analizar en detalle en cuáles tareas se desempeña mejor el dispositivo.

Para el CPU, hay tres pruebas: evaluación matemática, uso normal y evaluación multi-core.

Para el GPU, se evalúa la fluidez de gráficos en tres escenarios 3D.

Se incluye una prueba para medir el rendimiento de la experiencia del usuario, que incluye pruebas de procesamiento de imágenes, de datos y de la interfaz, como "scrolling cache".

Y por último, hay dos pruebas de la memoria: RAM y ROM.

En base a esto, cada categoría arroja una puntuación y la suma total sirve como medida de comparación global con otros dispositivos.

Para poder establecer una referencia de rendimiento del dispositivo, se hicieron dos ejecuciones del mismo benchmark. Para el primer test, se hizo la prueba con la versión N6F27M, con el kernel proveído por Google que deriva de la versión 3.10.40 y su fecha de compilación es el 16 de agosto de 2017. No fue modificada ninguna configuración del kernel.

La segunda prueba fue ejecutada en el mismo terminal con la versión de julio de 2018 de la distribución Nitrogen OS, cuya versión de Android corresponde a la versión 8.1 del

5 de julio de 2018, y cuyo kernel es ElementalX 5.22, que se basa en la versión 3.10.108 de Linux. Para realizar esta prueba, se hicieron cambios en algunas configuraciones:

CPU governor: Interactive, frecuencia máxima de 2,649MHz con los 4 núcleos activados.

GPU governor: Performance, frecuencia máxima de 600MHz.

Planificador I/O: noop

Readhead kb: 1024.

En ambos casos, había la misma cantidad de aplicaciones instaladas desde la tienda de aplicaciones, aunque en la versión oficial se incluyen alrededor de 15 aplicaciones extras de servicios de Google (las mismas que en la versión *stock* de Opengapps. Los resultados son mostrados en la figura 4.24).

De los benchmarks podemos decir que:

→ La puntuación mejoró por 4,000 puntos en el apartado de CPU.

→ Se perdió ligeramente en el apartado de rendimiento de gráficos por 300 puntos.

→ Se ganó sensiblemente en la experiencia del usuario por casi 3,000 puntos.

→ En cuanto al rendimiento de la memoria, mejoró por 1,000 puntos, pero representa un 20 % más.

→ Estos resultados eran esperados, pues el desarrollador del custom kernel cambia ciertos valores además de los modificados por nosotros, lo que en este caso favoreció un mejor performance en los benchmarks. Por lo general, los fabricantes establecen sus configuraciones para que el dispositivo ofrezca un desempeño equilibrado en todos los escenarios, permitiendo una autonomía de batería aceptable, y manteniendo la temperatura del dispositivo dentro de un margen seguro. Al modificar valores del kernel para obtener mayor rendimiento, la autonomía de la

Figura 4.24: Puntuaciones: arriba distribución stock versión 7.1.1, abajo custom ROM de Android 8.1.

batería se verá afectada por el mayor uso de energía, además de que el dispositivo podría enfrentarse a temperaturas elevadas, que en primera instancia supondría una merma de rendimiento pues el kernel se encargará de ralentizar al CPU, GPU y memoria, y en algunas ocasiones el rendimiento se verá afectado negativamente, en lugar de mejorar.

→ Las mayores puntuaciones recaen directamente en el uso de un gobernador del GPU más agresivo, lo cual permitió que en algunas pruebas el dispositivo rindiera mejor, aunque la prueba dedicada a medir el performance directo de la GPU arrojó

valores ligeramente inferiores.

→ Por último, una nueva versión del sistema operativo ayuda en algunas tareas. Los desarrolladores oficiales de Android han optimizado elementos claves en el rendimiento, como ejemplo se puede mencionar al recolector de basura, el cual en cada versión de Android ha sido cada vez más eficiente.

Es posible enunciar que los resultados, sin ser excepcionales, permiten aseverar que el rendimiento puede ser modificado con la ayuda de la sustitución de otra distribución de Android y el uso de parámetros personalizables por medio de un kernel modificado por la comunidad abierta.

4.4. Estadísticas de uso de distribuciones de la comunidad abierta

De las distribuciones de la comunidad abierta, la que mejor aceptación y soporte que sus desarrolladores proporcionan es LineageOS. De acuerdo al sitio de internet `https://stats.lineageos.org/` hay más de 1,812,000 instalaciones activas en dispositivos compatibles con esta distribución. México ocupa el noveno lugar con más de 35,000 instalaciones activas. En cuanto al dispositivo Motorola Nexus 6, ocupa el lugar 24 con más de 14,500 instalaciones.

La última versión de la distribución utilizada en esta tesis ha sido descargada únicamente 250 veces en su versión para el Nexus 6, evidenciando que es una distribución con baja cuota de usuarios, y cuyo equipo de desarrollo es limitado.

4.5. Conclusiones

Podemos concluir que:

1. El remplazo del firmware original por uno desarrollado por la comunidad abierta permite que el dispositivo obtenga actualizaciones de software aún cuando el terminal ya no tiene soporte oficial, lo que incrementaría la seguridad del usuario en comparación con la última versión oficial.

2. Un usuario podría extender la vida útil de un teléfono usando una distribución de la comunidad abierta, siempre y cuando haya algún desarrollador o equipo de la comunidad abierta que siga soportando dicho dispositivo.

3. Una distribución de la comunidad abierta aportará una mayor cantidad de configuraciones que el usuario puede personalizar. Además al obtener permisos "root" en el dispositivo podremos modificar valores del kernel que antes no podían ser modificados.

4. Las distribuciones de la comunidad abierta no ofrecen una experiencia totalmente libre de errores, hay ocasiones en las que el dispositivo no funciona correctamente.

5. Las distribuciones de la comunidad abierta suelen obtener actualizaciones con mayor frecuencia que las oficiales, ya sea porque son compiladas a partir de las versiones que Google lanza mensualmente, o porque la comunidad abierta se encarga de añadir nuevas características sin necesidad de tratarse de una versión diferente del sistema operativo.

6. Las distribuciones de la comunidad abierta ofrecen la posibilidad de ejecutar versiones nuevas de Android en dispositivos cuyo fabricante no contempló hacerlo, principalmente porque el soporte del equipo oficial finalizó, o porque el fabricante no contempló actualizarlo, dejándolo obsoleto.

7. Ningún proyecto desde la comunidad abierta es infalible. Hay proyectos que ofrecen mejor estabilidad que otros, pero no se puede garantizar que el proyecta sea

libre de errores.

Bibliografía

[1] Slim roms. Consultado el 3 de mayo de 2018 en https://slimroms.org/.

[2] Tuning virtual memory. Consultado el 15 de junio de 2018 en https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/s-memory-tunables.

[3] Root definition, 2005. http://www.linfo.org/root.html.

[4] Privacy policy, 2016. http://www.supersu.com/about#privacy-policy-supersu.

[5] Android, 2018. Consultado el 4 de mayo de 2018 en https://wiki.archlinux.org/index.php/android.

[6] Android debug bridge, 2018. Consultado el 28 de junio de 2018 en https://developer.android.com/studio/command-line/adb.html.

[7] Improving performance, 2018. Consultado el 15 de junio de 2018 en https://wiki.archlinux.org/index.php/improving_performance #Input.2Foutput_schedulers.

[8] Tuning the memory management subsystem, 2018. https://doc.opensuse.org/documentation/leap/tuning/html/book.sle.tuning/ .tuning.memory.html.

[9] Dominik Brodowski. Cpu frequency and voltage scaling code in the linux(tm) kernel. Consultado el 3 de junio de 2018 en https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt.

[10] Neil Brown. An f2fs teardown, 2012. https://lwn.net/Articles/518988/.

[11] John Callaham. The history of android os: its name, origin and more, 2018. https://www.androidauthority.com/history-android-os-name-789433/.

[12] Ben Francis. The story of firefox os, 2017. https://medium.com/@bfrancis/the-story-of-firefox-os-cb5bf796e8fb.

[13] M. Tim Jones. Anatomy of the linux kernel, 2007. https://www.ibm.com/developerworks/library/l-linux-kernel/.

[14] Steve Kondik. A new chapter, 2013. http://www.cyanogenmod.org/blog/a_new_chapter.

[15] "Koush". How-to-root, hack, and flashing your g1/dream, 2008. https://forum.xda-developers.com/showthread.php?t=442480.

[16] Thomas Krenn. Linux page cache basics, 2013. https://www.thomas-krenn.com/en/wiki/Linux_Page_Cache_Basics.

[17] Jonathan Levin. *Android Internals - A Confectioner's Cookbook - Volume I - The Power User's View.* Technologeeks.com, 2015.

[18] linfo.org. Root filesystem definition, 2006. http://www.linfo.org/root_filesystem.html.

[19] Robert Love. *Linux system programming.* 2013.

[20] Stefano Mazzocchi. Dalvik: how's google routed around sun's ip-based licensing restrictions on java me, 2007. https://web.archive.org/web/20110225035845/http://www.betaversion.org/ stefano/linotype/news/110/.

[21] Robert McMillan. A jailbreak for google's android, 2008. https://www.pcworld.com/article/153387/article.html.

[22] Falcon Momot. Some questions about kernel.random.* parameters, 2014. https://serverfault.com/questions/605275/some-questions-about-kernel-random-parameters.

[23] Iván Ramírez. Chainfire se retira oficialmente de supersu, pero su desarrollo continúa, 2017. https://www.xatakandroid.com/roms-android/chainfire-se-retira-oficialmente-de-supersu-pero-su-desarrollo-continua.

[24] Vicknesh Rethinavelu. What is tcp congestion algorithm in kernel of android?, 2017. https://www.quora.com/What-is-TCP-Congestion-Algorithm-in-Kernel-of-Android.

[25] Cammeron Summerson. What is "systemless root" on android, and why is it better?, 2017. https://www.howtogeek.com/249162/what-is-systemless-root-on-android-and-why-is-it-better/.