

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Facultat d'Informàtica de Barcelona (FIB)

---

# Creación de un laboratorio virtual para el aprendizaje de sistemas operativos utilizando Android

---



Grado en Ingeniería Informática

Tecnologías de la Información

Autor: Aitor ARDILA Bellés

Director: Enric MORANCHO Llena

*Arquitectura de computadores*

13 de enero de 2017

*“ Android is one of the most open systems I’ve ever seen. What makes Android great is it’s literally designed from the ground up to be customised in a very powerful way. ”*

*Sundar Pichai - CEO of Google Inc.*

# Resumen

Android es el sistema operativo móvil más usado actualmente. Por tanto, los estudiantes de sistemas operativos deberían ser capaces de entender cómo funciona Android por dentro y disponer de un entorno de trabajo adaptado a ello.

Este proyecto pretende adaptar la asignatura de Sistemas Operativos creando un laboratorio virtual con todas las herramientas necesarias para trabajar con el código fuente de Android, implementándole una serie de mejoras. Se analizan distintas herramientas y se hace un estudio comparativo para determinar la que mejor se adapta a las necesidades del proyecto. Además, se proponen e implementan una serie de prácticas de laboratorio enfocadas a tratar distintas partes del kernel de Android.

# Resum

Android és el sistema operatiu mòbil més usat actualment. Per tant, els estudiants de sistemes operatius haurien de ser capaços d'entendre com funciona Android per dins i disposar d'un entorn de treball adaptat a ell.

Aquest projecte pretén adaptar l'assignatura de Sistemes Operatius creant un laboratori virtual amb totes les eines necessàries per treballar amb el codi font d'Android, implementant-hi una serie de millores. S'analitzen diferents eines i es realitza un estudi comparatiu per determinar la que millor s'adapta a les necessitats del projecte. A més, es proposen i implementen una serie de pràctiques de laboratori enfocades a tractar les diferents parts del kernel d'Android.

# Abstract

Android is the most used mobile operating system. Therefore, the pupils studying Operating Systems subject should be able to understand how Android works from the inside and also provide them a work environment to work with Android.

The aim of this project is to try to adapt the Operating Systems subject by creating a virtual lab with all the necessary tools for working with the Android source code and implementing some improvements to this lab. Some tools are analyzed and a comparative study is done in order to determine what is the best tool that adapts better to the needs of this project. Moreover, some lab tasks are implemented so that the students can work on different parts of the Android kernel.

# Índice general

<b>1. Contexto</b>	<b>1</b>
1.1. Objeto de estudio . . . . .	1
1.2. Actores implicados . . . . .	2
1.3. Formulación del problema . . . . .	3
<b>2. Estado del arte</b>	<b>5</b>
<b>3. Alcance del proyecto</b>	<b>8</b>
3.1. Definición del alcance . . . . .	8
3.2. Posibles obstáculos y planes de contingencia . . . . .	9
3.3. Metodología de trabajo . . . . .	10
3.4. Herramientas de seguimiento . . . . .	11
3.5. Métodos de validación . . . . .	11
3.6. Evaluación final . . . . .	12
<b>4. Planificación temporal</b>	<b>13</b>
4.1. Planificación general . . . . .	13
4.1.1. Duración del proyecto . . . . .	13
4.1.2. Recursos . . . . .	13
4.1.3. Valoración de alternativas y plan de acción . . . . .	15
4.2. Descripción de las tareas . . . . .	15
4.2.1. Estimación de horas por tarea . . . . .	16
4.2.2. Diagrama de Gantt . . . . .	18

---

<b>5. Gestión económica</b>	<b>21</b>
5.1. Consideraciones generales . . . . .	21
5.2. Identificación y estimación de los costes . . . . .	21
5.2.1. Costes directos por actividad . . . . .	21
5.2.2. Costes indirectos . . . . .	22
5.2.3. Contingencia . . . . .	25
5.2.4. Imprevistos . . . . .	25
5.2.5. Presupuesto . . . . .	25
5.3. Control de gestión . . . . .	26
<b>6. Background de Android</b>	<b>28</b>
6.1. Características principales . . . . .	28
6.2. Android Open Source Project . . . . .	29
6.3. Diferencias entre el kernel de Android y Linux . . . . .	29
6.4. Proceso de boot . . . . .	31
6.5. Particiones . . . . .	32
6.6. Sistema de ficheros . . . . .	34
<b>7. Laboratorio virtual</b>	<b>36</b>
7.1. Consideraciones generales . . . . .	36
7.2. Análisis de requisitos . . . . .	36
7.2.1. De hardware . . . . .	36
7.2.2. De software . . . . .	37
7.3. Dispositivo físico . . . . .	37
7.4. Emuladores . . . . .	40
7.4.1. Comparativa entre emuladores . . . . .	42
7.4.2. Tests de rendimiento . . . . .	43
7.4.3. Conclusiones . . . . .	45
7.5. Código fuente . . . . .	47
7.5.1. Descarga . . . . .	47
7.5.2. Compilación . . . . .	48

---

7.5.3. Ejecución . . . . .	51
7.6. Mejoras implementadas . . . . .	53
7.6.1. Cross-compiling . . . . .	53
7.6.2. Script de automatización . . . . .	54
7.7. Creación del laboratorio virtual . . . . .	56
7.7.1. Sugerencia 1: Imagen de Ubuntu . . . . .	57
7.7.2. Sugerencia 2: Guía de instalación . . . . .	58
7.7.3. Sugerencia 3: Guía de instalación con código fuente . . . . .	58
7.7.4. Evaluación . . . . .	59
<b>8. Prácticas de laboratorio</b>	<b>61</b>
8.1. Consideraciones generales . . . . .	61
8.2. Práctica 1: Instalación del entorno virtual . . . . .	62
8.3. Práctica 2: Governor de Android . . . . .	62
8.4. Práctica 3: Añadir app de sistema . . . . .	64
8.4.1. Aplicación Android . . . . .	65
8.4.2. Programa en C . . . . .	68
8.5. Práctica 4: Crear módulo de kernel . . . . .	70
8.6. Práctica 5: Sistema de ficheros F2FS . . . . .	72
8.7. Análisis valorativo . . . . .	74
<b>9. Sostenibilidad y compromiso social</b>	<b>75</b>
9.1. Ámbito económico . . . . .	75
9.2. Ámbito social . . . . .	75
9.3. Ámbito ambiental . . . . .	76
9.4. Matriz de sostenibilidad . . . . .	76
<b>10. Conclusiones</b>	<b>77</b>
<b>Bibliografía</b>	<b>79</b>



# Índice de figuras

1.1. Número total de usuarios de <i>smartphone</i> desde 2013 a 2018 [3] . . . . .	2
2.1. Resultados de la encuesta del curso de SO de la Universidad de Columbia [5]	6
3.1. Distintas fases del proyecto . . . . .	11
4.1. Planificación de las tareas y su riesgo . . . . .	19
4.2. Diagrama de Gantt . . . . .	20
6.1. Logo oficial de Android . . . . .	28
6.2. Nombre, versión y API de Android desde la versión 4.4 a la actual 7.1. . .	29
6.3. Funciones de la API de Wakelocks en el espacio del kernel. . . . .	30
6.4. Esquema con algunas de las comunicaciones disponibles en Binder. . . . .	30
6.5. Menú principal, modo instalación y modo shell del Recovery TWRP 3.0-2-0.	33
6.6. Particiones montadas en el dispositivo Motorola LTE E. . . . .	34
6.7. Particiones existentes en el dispositivo Motorola LTE E. . . . .	35
7.1. Captura de pantalla de CyanogenMod . . . . .	39
7.2. Captura de pantalla del Android Emulator . . . . .	40
7.3. Captura de pantalla del Genymotion . . . . .	41
7.4. Tiempo de arranque de Android de cada emulador (segundos) . . . . .	43
7.5. Consumo de CPU en la pantalla inicial de Android de cada emulador (%).	44
7.6. Consumo de Memoria RAM en la pantalla inicial de Android de cada emulador (MB). . . . .	44
7.7. Resultados de AnTuTu Benchmark divididos en 3D, UX, CPU y RAM por emulador. . . . .	45

---

7.8. Puntuación total de AnTuTu Benchmark de cada emulador . . . . .	46
7.9. Tamaño que ocupa en disco el código fuente de CyanogenMod. . . . .	48
7.10. Comando de compilación en make con 8 núcleos . . . . .	50
7.11. Salida del comando <i>htop</i> durante la compilación de Android. . . . .	50
7.12. Contenido del directorio <i>out/target/product/surnia</i> . . . . .	51
7.13. Pantalla de Wipe con el programa de recovery TWRP. . . . .	52
7.14. Ejecución de HelloWorld compilado para ARM en dispositivo físico mediante <i>adb shell</i> . . . . .	53
7.15. Mensaje de error mostrado al no haber inicializado ADB en modo root previamente. . . . .	55
7.16. Ejecución correcta del script tras inicialización de ADB en modo root. . . . .	55
7.17. Menú principal del software Clonezilla Live . . . . .	57
7.18. Progreso de clonación para un dispositivo de 5GB . . . . .	58
8.1. Directorio con todos los ficheros <i>governator</i> para Surnia. . . . .	63
8.2. Ejecución exitosa de <i>adb root</i> y <i>adb devices</i> . . . . .	65
8.3. Captura de pantalla donde se observa la aplicación <i>Remote Control</i> instalada. 66	
8.4. Contenido del directorio <i>packages/apps</i> que forma parte del código fuente de CyanogenMod 13.0. . . . .	67
8.5. Copia y ejecución del programa en C en el directorio <i>/system/bin</i> de Android. 70	
8.6. Salida del comando <i>dmseg</i> donde se observa el string Hello World. . . . .	72

# Índice de tablas

4.1. Recursos Software . . . . .	14
4.2. Recursos Hardware . . . . .	14
4.3. Fechas previstas de finalización y defensa oral del proyecto. . . . .	15
4.4. Horas por tarea (1/2) . . . . .	17
4.5. Horas por tarea (2/2) . . . . .	18
5.1. Coste estimado de cada rol . . . . .	22
5.2. Costes directos por actividad (1/2) . . . . .	23
5.3. Costes directos por actividad (2/2) . . . . .	24
5.4. Costes indirectos . . . . .	24
5.5. Contingencia . . . . .	25
5.6. Coste de los imprevistos . . . . .	26
5.7. Presupuesto . . . . .	26
7.1. Requerimientos hardware . . . . .	37
7.2. Requerimientos software para la compilación del código fuente de Android 6.0	38
7.3. Requerimientos software para el desarrollo, comunicación y depuración de aplicaciones Android 6.0 . . . . .	38
7.4. Comparativa de las características de cada emulador . . . . .	42
8.1. Resultados de ejecución de todos los benchmark usando EXT4 y F2FS. . .	74
8.2. Número de horas de dedicación, dificultad de cada práctica de laboratorio y curso recomendado. . . . .	74
9.1. Matriz de sostenibilidad . . . . .	76

# Capítulo 1

## Contexto

### 1.1. Objeto de estudio

Este proyecto es un Trabajo Final de Grado (TFG) llevado a cabo en la Facultad de Informática de Barcelona (FIB) la cual forma parte de la Universidad Politécnica de Cataluña (UPC). El autor de este proyecto pertenece a la especialidad de Tecnologías de la Información.

El proyecto en sí nace de la posibilidad de adaptar la asignatura de Sistemas Operativos impartida en la FIB, debido al amplio uso de los dispositivos portátiles. Hasta ahora, se ha estado usando *Linux* [1] como modelo para introducir a los estudiantes a los Sistemas Operativos. A partir de la versión de escritorio, se forma a los estudiantes en las distintas características de este sistema operativo y sus funcionalidades. El estudiante, además, realiza una serie de proyectos en el laboratorio para demostrar que es capaz de aplicar los conocimientos adquiridos a lo largo del curso.

Esta fórmula ha funcionado bien hasta ahora. Pero desde hace unos años los sistemas operativos de escritorio han perdido importancia en el sector de la informática, debido al auge de los dispositivos móviles: *smartphones*, *tablets*, *smartwatches*, *etc* . El panorama de la computación está evolucionando hacia los dispositivos móviles, y éstos van en aumento año tras año. En la Figura 1.1 se puede observar este incremento. Debido a esta evolución, el profesor de Sistemas Operativos y director de este proyecto, Enric Morancho Llena, consideró la posibilidad de incluir *Android* [2] como modelo de estudio de los sistemas operativos, en vez de centrarse únicamente en un sistema Linux de escritorio.

Con este proyecto, se ha procurado trabajar sobre un sistema de uso cotidiano como es Android y que además tuviera una finalidad práctica, a parte de todo el aprendizaje teórico

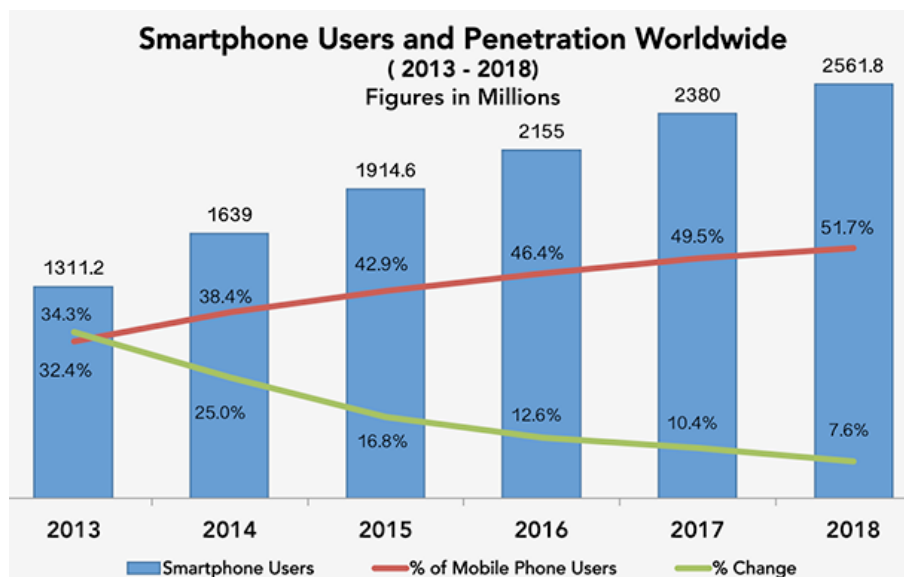


Figura 1.1: Número total de usuarios de *smartphone* desde 2013 a 2018 [3]

que hay detrás. Se ha buscado desarrollar un entorno de desarrollo (laboratorio virtual) con todas las herramientas necesarias para que los estudiantes sean capaces de trabajar con Android, de una manera cómoda y eficaz. Una vez el entorno está disponible, se han propuesto una serie de prácticas de laboratorio usando este entorno.

El laboratorio virtual que se ha desarrollado contiene todas las herramientas necesarias para poder editar, compilar, depurar y ejecutar el código para Android. Además, el código puede ser ejecutado tanto en un dispositivo físico (ej. un *smartphone*) como en uno de virtual (ej. el emulador del Android SDK llamado *Android Virtual Device* [4]). Al finalizar el curso, el estudiante debería ser capaz de entender los principales componentes de Android y qué aportan de nuevo respecto a un sistema Linux tradicional de escritorio, hacer pequeños cambios en el kernel de Android y por supuesto ser capaz de compilar el código fuente y hacerlo funcionar en un dispositivo real.

## 1.2. Actores implicados

A continuación se enumeran los distintos actores implicados (directa o indirectamente) en el proyecto:

- **Desarrollador:** Es la persona principal del proyecto, ya que es la encargada de llevar a cabo la implementación y documentación del mismo. Sus principales funciones son: desarrollar el laboratorio virtual de Android y comprobar su correcto funcionamiento,

documentar todo el trabajo realizado y que quede reflejado en la memoria final, y preparar una presentación final. Además, debe estar en contacto continuo con el director del proyecto y procurar cumplir con los plazos de tiempo y objetivos pactados inicialmente.

- **Director:** Es la persona que propone el proyecto y que define el planteamiento inicial. En este proyecto, el profesor Enric Morancho Llena tiene el rol de director. Su rol es también muy importante ya que es quien aconseja al desarrollador, le aporta fuentes de información e ideas que son de gran utilidad para el desarrollo del proyecto.
- **Estudiantes:** Es el colectivo de personas que harán un uso práctico de este proyecto. Tienen por tanto, el rol de beneficiarios. Serán los usuarios finales del laboratorio virtual de Android, usarán el entorno de trabajo implementado en el proyecto y realizarán las prácticas propuestas en las clases de laboratorio.

### 1.3. Formulación del problema

La realización de este proyecto tiene dos objetivos principales:

- **Desarrollo de un laboratorio virtual:** Implementar un entorno virtual con todas las herramientas de software necesarias para poder trabajar en Android. Previamente a la implementación, se estudian las distintas herramientas disponibles y se escoge la que mejor se adapta a las necesidades de la asignatura, siguiendo una serie de criterios a evaluar. Entre otros, el laboratorio virtual contiene:
  - SDK de Android
  - Android Studio
  - Código fuente de Android
  - Emulador virtual de Android
  - *Toolchains*<sup>1</sup> para compilar Android
  - Herramientas de depuración de código
  - Herramientas de *cross-compiling*
- **Propuesta de prácticas de laboratorio:** Una vez el laboratorio virtual ha sido implementado y está operativo, se proponen una serie de prácticas para que los futuros estudiantes puedan aplicar de manera práctica los conocimientos teóricos adquiridos en clase. Estas prácticas se centran en conceptos importantes del SO y más en concreto aplicados a Android. Básicamente consisten en modificar el kernel

---

<sup>1</sup>Conjunto de herramientas necesarias para editar, compilar y depurar código

de Android, compilarlo y comprobar su correcto funcionamiento. Las prácticas que se pronen realizar son:

- Instalación del laboratorio virtual
- Modificar el *governor* de Android
- Añadir una aplicación de sistema
- Crear un módulo de kernel
- Sistema de ficheros F2FS

# Capítulo 2

## Estado del arte

El estudio de los sistemas operativos se ha venido impartiendo en las universidades desde el surgimiento de los sistemas operativos. Asimismo, desde que se lanzó Android al mercado, las universidades se han adaptado y muchas de ellas han incluido el estudio de Android en su temario. En la Universidad Politécnica de Cataluña (UPC) se imparte Android en asignaturas como:

- Interacción y diseño de Interfaces (IDI): Se construye una aplicación para Android.
- Conceptos Avanzados de Sistemas Operativos (CASO): Se estudian las diferencias entre un SO de sobremesa/portátil con otro que da soporte a dispositivos móviles (Android).
- Internet Móvil (IM): Se describen las características y funcionalidades más importantes de Android como SO para dispositivos móviles.

Ninguna de estas asignaturas tiene un laboratorio virtual desde el cual poder trabajar con el kernel de Android. De entre las tres, CASO es la asignatura que más se aproxima en la vertiente teórica, pero no en la práctica, al no disponer del laboratorio virtual.

En cuanto al resto de universidades, cabe señalar que los programas educativos de muchas universidades no son de libre acceso. De manera que la forma de saber el estado del arte es realizando una búsqueda de proyectos similares, artículos, etc. Existe un proyecto realizado en el año 2012 por dos profesores del departamento de *Computer Science* de la universidad de Columbia (NY). En el artículo que publicaron, titulado *Teaching Operating Systems Using Android [5]* se explica cómo ambos profesores desarrollaron su propio entorno virtual para introducir a sus estudiantes en el mundo de Android desde el punto de vista de los sistemas operativos. Sin dar muchos detalles, exponen en qué se basa su sistema. En resumidas cuentas, crearon una imagen virtual de Linux con todas las herramientas



necesarias para trabajar con el kernel de Android, a través de prácticas de laboratorio. Su proyecto comparte muchas características con este, aunque no dan detalles explícitos de las herramientas y tecnologías usadas. Al finalizar el curso, se les realizó una encuesta a los estudiantes. El resultado, que puede verse en la Figura 2.1, fue muy satisfactorio.

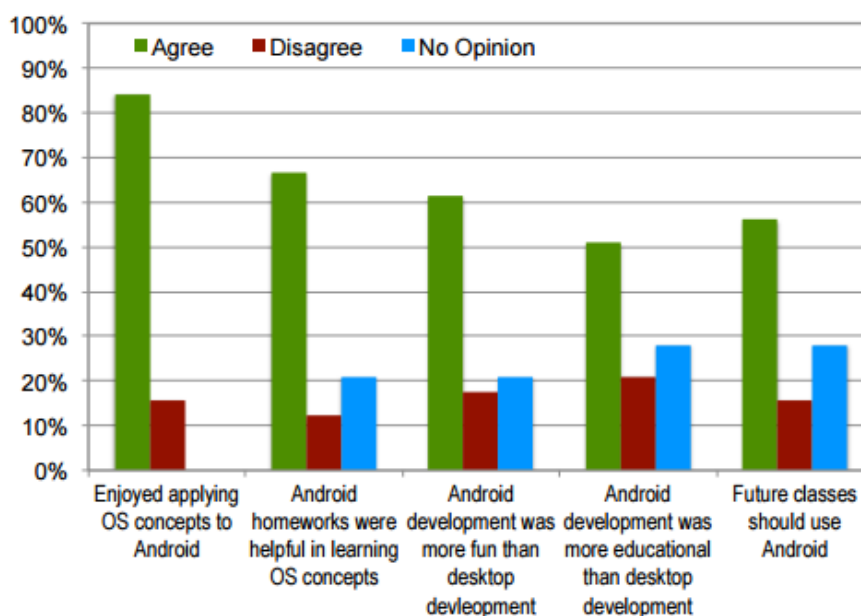


Figura 2.1: Resultados de la encuesta del curso de SO de la Universidad de Columbia [5]

En los resultados de búsqueda de proyectos similares de Android, solo se han encontrado de la Universidad de Columbia. No se han encontrado publicaciones de otras universidades o centros que impartan SO a partir de Android. Sí que lo mencionan (comparándolo con otros SO de escritorio, por ejemplo), pero sin llegar a trabajar con él de la manera en que se ha perseguido con este proyecto.

A parte de Android, existen otros proyectos educativos ya implementados que aunque no usen Android comparten una idea similar con la de este proyecto: adaptar la asignatura de Sistemas Operativos a los nuevos cambios tecnológicos. Por ejemplo, en la facultad de Ingeniería Eléctrica de la Universidad de Ljubljana implementaron para la asignatura la enseñanza de los sistemas embebidos [6] actuales al ver el auge de estos sistemas en el sector tecnológico [7].

Otros proyectos existentes no se centran en el contenido sino en la metodología de aprendizaje. Por ejemplo, en el Instituto Tecnológico de Messolonghi (Grecia) implementaron un software interactivo basado en la web para enseñar Sistemas Operativos [8]. También la Universidad de Postdam (Alemania) implementó como mejora de la asignatura un proyecto

de software que es capaz de medir diversas métricas del kernel de Linux [9].

Tras nuestro estudio podemos concluir que, aparentemente, a día de hoy la enseñanza de SO a través de Android no está implementada salvo en alguna universidad como la de Columbia. Con esto, este proyecto puede llegar a adaptar la asignatura de Sistemas Operativos a los nuevos dispositivos y, si obtienen buenos resultados, ser usado como modelo para otras universidades o centros educativos.

# Capítulo 3

## Alcance del proyecto

### 3.1. Definición del alcance

El sistema resultante de este proyecto debe incluir:

- **Laboratorio virtual:** Se trata de desarrollar un entorno a partir de *Ubuntu* [10] que incluya todo el software y herramientas necesarias para poder trabajar con el kernel de Android. El listado de herramientas necesarias puede observarse en el apartado 1.3. Todo el software que contiene es libre y no requiere licencia. Para testear el resultado de las implementaciones en el kernel, el laboratorio virtual dispone de:
  - **Emulador de Android:** Se trata de un emulador de un dispositivo Android capaz de arrancar el SO y las modificaciones que se le realicen tanto del kernel como de sus aplicaciones. Además, es capaz de ejecutar programas en C compilados previamente desde el PC, usando para ello las herramientas de *cross-compiling*<sup>1</sup> incluidas en el laboratorio virtual. También se pueden transferir datos desde el host (PC) al emulador y a la inversa.
  - **Dispositivo físico:** El laboratorio virtual es capaz de comunicarse también con un dispositivo físico preparado previamente para poder ser usado para este sistema. Para ello, el teléfono debe estar *rootado* y tal que el modo *recovery* acepte la instalación de otra imagen de Android. Para las pruebas de testing en este proyecto, se ha usado como dispositivo físico un smartphone *Motorola E LTE*. Este terminal está preparado para ser *flasheado* con una nueva imagen de Android. También puede ejecutar programas en C (después de que el código haya sido compilado para arquitectura ARM) y es capaz de ser controlado por consola desde el host (PC) conectado mediante cable USB.

---

<sup>1</sup>Es necesario debido a que el programa se ejecuta en un arquitectura ARM vs la x86 del PC

- **Listado de prácticas de laboratorio:** Contiene un listado con algunas propuestas de prácticas de laboratorio, cada una de las cuales se centrará en una parte del sistema operativo. Los ámbitos del SO en los cuales se basarán las prácticas pueden consultarse en el apartado 1.3. Estas prácticas son de un nivel asequible para un estudiante de la asignatura de Sistemas Operativos y deberían ser de gran utilidad para poner en práctica todo los conocimientos teóricos de Android adquiridos en clase.

## 3.2. Posibles obstáculos y planes de contingencia

Los posibles obstáculos que pueden surgir durante la realización de este proyecto son los siguientes:

- **Duración del proyecto:** El Trabajo de Final de Grado tiene una duración con una fecha límite. Por lo tanto, hay que procurar seguir el guión de trabajo planteado inicialmente y cumplir con las fechas marcadas. Si hay riesgo de retraso durante la fase de desarrollo, hay que gestionarlo adecuadamente para que no afecte a la entrega final. Por ejemplo, replanteándose un cambio no previsto de ese punto del proyecto o disponer de un plan alternativo para corregirlo a tiempo.
- **Resultado abierto:** En la fase inicial de desarrollo es difícil plantear de antemano todos los detalles del resultado final. Sí que se planea crear un laboratorio virtual y propuestas de prácticas de laboratorio, pero los detalles de cómo será no se pueden prever hasta que el desarrollo del proyecto no avance. Para solventar este inconveniente, es muy importante tener muy claro el objetivo global y documentarse muy bien de qué se puede modificar de Android, cómo y con qué herramientas.
- **Emulador de Android:** Es posible que no se encuentre un emulador virtual que cumpla con todos los requisitos del proyecto. Por ejemplo, que no permita *flashear* una imagen propia o que sí que lo permita pero solo usando una versión *premium*. De no disponer de ningún emulador, se usaría el emulador por defecto de Android Studio: *Android Virtual Device (AVD)*, aunque éste usa una arquitectura ARM (igual que un terminal físico), la cual al ser emulada en el PC, que tiene arquitectura x86, ralentiza la ejecución del SO y las aplicaciones. Dejando de lado la lentitud, este emulador sigue cumpliendo con todos los requerimientos, además de ser *freeware*, por lo que, si no hay otra alternativa, podría tener cabida en el laboratorio virtual.
- **Resultado práctico:** Al finalizar el proyecto, es posible que éste no llegue a usarse nunca con fines prácticos, porque se ve que no es aplicable en la asignatura de Sistemas Operativos. También cabe la posibilidad que sí se pruebe una vez en la asignatura, pero que no tenga el éxito esperado y decida no usarse más. Para evitar esto, aunque

siempre exista una probabilidad de fracaso, se realizan reuniones con el director del proyecto para estudiar cómo debe ser el resultado final.

- **Nivel requerido:** Hay que tener en cuenta que el laboratorio virtual y las prácticas serán usadas por estudiantes de segundo de Grado, por lo que el nivel técnico debe ser el adecuado. La preparación del entorno de trabajo debe estar lo suficientemente bien documentado para que cualquier estudiante pueda implementarlo de manera autónoma. También las prácticas deben ser acorde al nivel requerido de la asignatura. Por lo tanto, hay que tener este punto muy en cuenta en todo momento.
- **Pérdida de información:** Para prevenir posibles errores hardware, errores humanos o catástrofes que puedan causar la pérdida de datos, se tiene una copia en la nube, ya que los datos no son confidenciales. En concreto, se almacena todo en *Google Drive*. Para la redacción de la memoria final, se emplea el servicio online *ShareLaTeX* [13], basado en *LaTeX* [12], que al estar en la nube permite tener un backup implícito de la documentación.

### 3.3. Metodología de trabajo

La metodología de trabajo está dividida en cuatro fases principales, las cuales siguen una metodología de cascada:

1. **Fase inicial:** Conformar la fase donde se realiza el curso de GEP.
2. **Fase de adquisición de la información:** Se recopila toda la información posible para poder empezar a implementar el laboratorio virtual.
3. **Fase de implementación:** Se divide en 2 tareas principales: la implementación del laboratorio virtual y la elaboración de las prácticas de laboratorio. Cada tarea contendrá subtareas las cuales seguirán una metodología Scrum, es decir, partirán de unos requisitos iniciales, a continuación se implementarán y por último se realizará una evaluación final [11].
4. **Fase final:** Conformar las tareas de evaluación del resultado final, revisión de la Memoria y una presentación final oral.

Paralelamente a las cuatro fases, a medida que se vayan completando las tareas y subtareas, se irán documentando sus resultados en la memoria final del proyecto. En la Figura 3.1 puede observarse el esquema de las cuatro fases.

#	FASES	TAREAS
1	inicial	GEP
2	adquisición información	Recopilación información
3	implementación	Laboratorio Virtual Prácticas laboratorio
4	final	Evaluación resultado final Revisión documentación Presentación final

Documentación  
Memoria

Figura 3.1: Distintas fases del proyecto

### 3.4. Herramientas de seguimiento

Las reuniones con el director sirven para llevar un control del estado actual del proyecto. Se planifican con antelación vía email o en persona. Estas reuniones se llevan a cabo cada 2 semanas (habitualmente) y suelen coincidir con la finalización de alguna subtarea o de algún logro o echo destacable de la fase de desarrollo. Con cada reunión se evalúan los resultados y resuelven dudas y se sugieren ideas o cambios para mejorar el resultado final.

Para llevar un control de estas reuniones, se ha acordado con el director rellenar una tabla con los siguientes campos: fecha de la reunión, motivo (tarea relacionada) y observaciones. De esta manera es posible tener un seguimiento más preciso de cada una de las reuniones.

### 3.5. Métodos de validación

Uno de los métodos es el explicado en el apartado 3.4 que consiste en las conclusiones extraídas de cada reunión con el director. El otro método de evaluación consiste en testear las modificaciones del kernel de Android. El testing se realizará tanto en el dispositivo físico como en el emulador virtual.

## **3.6. Evaluación final**

Para la evaluación final, el director del proyecto estudiará si el resultado final es el esperado y si es viable para ser usado en la asignatura de SO. Cumplir con los requisitos iniciales es indispensable para que sea considerado un buen resultado.

# Capítulo 4

## Planificación temporal

### 4.1. Planificación general

#### 4.1.1. Duración del proyecto

El tiempo aproximado para la realización del proyecto es de 4 meses, comprendidos entre septiembre de 2016 hasta enero de 2017. La carga aproximada de trabajo es de 450 horas, teniendo en cuenta que el proyecto equivale a 18 créditos ECTS donde cada crédito son 25 horas aproximadamente. Para tener una buena planificación del tiempo, se ha realizado una estimación de las horas necesarias para realizar cada una de las tareas. No obstante, se pueden producir desviaciones del tiempo previsto debido a algunos condicionantes que pueden alterar al proyecto, de manera que hay que tener en cuenta que ésta es una planificación temporal aproximada.

#### 4.1.2. Recursos

Para llevar a cabo cada una de las tareas, se disponen de una serie de recursos los cuales se pueden clasificar de la siguiente manera:

- **Recursos humanos:** Representa la figura del desarrollador<sup>1</sup>. Su especialidad es tecnologías de la información.
- **Recursos de software:** En la Tabla 4.1 se listan todos los recursos software necesarios.
- **Recursos de hardware:** En la Tabla 4.2 se listan los dos recursos hardware principales.

---

<sup>1</sup>En el apartado 1.2 se describe la figura del desarrollador.



Recurso	Tipo	Finalidad
Ubuntu 14.04 LTS	Desarrollo	SO desde el cual se desarrolla todo el proyecto.
Android Studio + SDK	Desarrollo	Para tener un entorno de programación de Android.
Gmail	Comunicación	Para comunicarse con el director del TFG.
Android Virtual Device	Desarrollo	Para emular Android en una arquitectura ARM.
Genymotion	Desarrollo	Para emular Android en una arquitectura x86.
CyanogenMod (código fuente)	Desarrollo	Código fuente desde el cual realizar las modificaciones.
Herramientas de cross-compiling	Desarrollo	Para compilar un programa en otra arquitectura.
Google Drive	Desarrollo	Para almacenar y compartir la documentación.
ShareLaTeX	Desarrollo	Para redactar la memoria final del proyecto.
Adobe Reader XI	Desarrollo	Para visualizar la documentación.
Ganttter	Desarrollo	Para construir el diagrama de Gantt.
Mendley	Desarrollo	Para formatear las referencias a artículos.

Tabla 4.1: Recursos Software

Recurso	Tipo	Finalidad
Portátil HP Notebook Envy 15-AK110NS i7-6700HQ	Desarrollo	Para desarrollar todo el proyecto.
Teléfono móvil inteligente Motorola E LTE	Desarrollo	Para testear las modificaciones de Android.

Tabla 4.2: Recursos Hardware

Finalización TFG	Holgura temporal	Defensa oral
10/01/2017	11/01/2017 - 24/01/2017	25/01/2017

Tabla 4.3: Fechas previstas de finalización y defensa oral del proyecto.

### 4.1.3. Valoración de alternativas y plan de acción

Durante la realización del proyecto, es posible que la planificación temporal de las tareas se vea alterada debido a cambios y/o contratiempos no previstos inicialmente. Debido a esto, se propone un plan de acción para solventar las posibles desviaciones temporales. Éste consiste en:

- **Reuniones con el director:** Las reuniones no solo sirven para ir al día de los avances del proyecto sino que cada reunión representa el fin de plazo de alguna tarea. De esta manera, se intenta finalizar la tarea antes del día de la reunión. En caso que por algún motivo no sea haya podido finalizar la tarea, se coordina con el director un aplazamiento de la reunión. El objetivo deseable es tener finalizada siempre la tarea antes de la reunión. En caso que no se cumpla con las fechas de entrega antes de la reunión, se estudiará junto con el director la mejor solución. Por ejemplo, un cambio en algún aspecto del proyecto.
- **Holgura temporal:** Debido al riesgo que existe de no tener finalizado el proyecto a tiempo, se propone un margen de dos semanas entre la fecha de finalización del proyecto y la fecha de la defensa oral. En la Tabla 4.3 están las fechas previstas de finalización y defensa oral del proyecto, junto con la holgura de dos semanas. En caso de que esto sucediera, no afectaría a ningún recurso ya que el dispositivo móvil no tiene una fecha de devolución pactada y en cuanto a los recursos software ninguno de ellos tiene caducidad. Sí afectaría, no obstante, al número de horas empleadas por el autor del proyecto en la realización del mismo.

## 4.2. Descripción de las tareas

En la sección 3.3 se describía la metodología de trabajo y se introducían las fases de las cuales consta el proyecto junto con las tareas asignadas a cada fase. Éstas son:

1. **Curso de GEP:** Corresponde a la fase inicial del proyecto. Durante cuatro semanas, se asignan una serie de tareas las cuales sirven para confeccionar los primeros capítulos de la memoria. Entre otros, se redacta el contexto, estado del arte, alcance, planificación temporal, gestión económica y sostenibilidad del proyecto. Al tratarse de la fase inicial, no tiene ninguna dependencia de precedencia. Aun así, las fechas límite de entrega son fijas, por lo que conviene no retrasarse en su elaboración.

2. **Recopilación de información:** Conforman la siguiente parte del proyecto. En ella, se debe buscar información que pueda ser útil de cara a que el desarrollador pueda resolver sus dudas, hacerse una idea del contexto en el cual trabaja y así poder empezar a implementar el proyecto. Conlleva también una parte de aprendizaje autónomo. Esta tarea tampoco tiene una dependencia de precedencia.
3. **Implementación del laboratorio virtual:** Forma parte de la fase de implementación. En ella, se desarrolla el entorno virtual desde el cual poder operar con Android. Las dependencias de precedencia son las dos anteriores. Esta tarea tiene asignadas una serie de subtareas, las cuales siguen un ciclo *Scrum*. Por cada subtarea, tras implementarla se prueba y una vez funcione se avanza con la siguiente subtarea.
4. **Elaboración de las prácticas de laboratorio:** Tiene como dependencia de precedencia todas las anteriores. En esta tarea, se elaboran una serie de prácticas de laboratorio de cara a que los estudiantes puedan practicar programando sobre el kernel de Android, usando para ello la plataforma virtual implementada anteriormente. Esta tarea tiene una serie de subtareas, cada una de las cuales representa una práctica de laboratorio. Las subtareas no son dependientes unas de otras, aún así, como solo hay un desarrollador se ha optado por seguir un orden secuencial.
5. **Documentación y presentación final:** Se trata de la fase final del proyecto y tiene como dependencia de precedencia todas las anteriores. Conforman las siguientes subtareas: evaluación del resultado final (posibles correcciones y conclusiones), revisión de la documentación y la presentación oral ante el tribunal. Todas tres siguen un orden secuencial.

A lo largo de todas las fases del proyecto, se realiza en paralelo la redacción de la memoria final. El mismo curso de GEP permite empezar a elaborar los primeros capítulos del documento. Durante la fase de implementación también se van documentando todos los avances. Por último, en la fase final, se organiza y revisa el documento final para ser entregado posteriormente.

#### 4.2.1. Estimación de horas por tarea

Se ha procurado hacer una partición del tiempo razonable según la carga de trabajo estimada de cada tarea. El total de horas es de aproximadamente 450 por lo que el reparto se ha realizado tomando como referencia este valor. En las Tablas 4.4 y 4.5 se han definido el reparto de horas por cada tarea.

Tarea	Responsable	Horas
<b>Curso de GEP</b>		<b>75</b>
Contexto, estado del arte y alcance	Jefe proyecto	24
Planificación temporal	Jefe proyecto	10
Gestión económica y sostenible	Jefe proyecto	10
Presentación preliminar	Jefe proyecto	7
Presentación oral y entrega final	Jefe proyecto	24
<b>Recopilación Información</b>		<b>30</b>
Búsqueda información	Jefe proyecto	15
Aprendizaje autónomo	Investigador	15
<b>Implementación lab. virtual</b>		<b>150</b>
Subtarea 1: Instalación del software		40
Análisis de requisitos	Analista	10
Instalación y configuración	Desarrollador	20
Testeo	Tester	10
Subtarea 2: Configuración del dispositivo móvil		20
Análisis de requisitos	Analista	5
Configuración	Desarrollador	10
Testeo	Tester	5
Subtarea 3: Compilación del código fuente Android y testeo en el dispositivo móvil.		20
Análisis de requisitos	Analista	5
Descarga, config. y compilación	Desarrollador	10
Testeo	Tester	5
Subtarea 4: empaquetamiento de todas las herramientas y creación del lab. virtual.		70
Análisis de requisitos	Analista	10
Empaquetamiento y creación lab.	Desarrollador 80 % Programador 20 %	50
Testeo	Tester	10

Tabla 4.4: Horas por tarea (1/2)

<b>Elaboración prácticas laboratorio</b>		<b>140</b>
Elaboración práctica 1	Desarrollador 75 %	28
	Programador 25 %	
Elaboración práctica 2	Desarrollador 75 %	28
	Programador 25 %	
Elaboración práctica 3	Desarrollador 75 %	28
	Programador 25 %	
Elaboración práctica 4	Desarrollador 75 %	28
	Programador 25 %	
Elaboración práctica 5	Desarrollador 75 %	28
	Programador 25 %	
<b>Documentación y presentación</b>		<b>55</b>
Documentación memoria	Jefe proyecto	40
Revisión final	Jefe proyecto	5
Presentación oral final	Jefe proyecto	10
<b>Total</b>		<b>450</b>

Tabla 4.5: Horas por tarea (2/2)

### 4.2.2. Diagrama de Gantt

Para realizar el diagrama de Gantt, se ha tenido en cuenta las dependencias entre las tareas y el tiempo previsto que hay que emplear para cada una de ellas. En la Figura 4.1 puede observarse el listado de las tareas con su fecha de inicio y fin. Cada color representa el riesgo de la tarea<sup>2</sup>. En la Figura 4.2 está representado el diagrama de Gantt.

<sup>2</sup>verde: riesgo bajo; naranja: riesgo medio; rojo: riesgo alto

Nombre	Duración	Inicio	Fin
<b>1 Curso de GEP</b>	21d	19/09/2016	17/10/2016
1.1 Contexto, estado del arte y alcance	7d	19/09/2016	27/09/2016
1.2 Planificación temporal	4d	28/09/2016	03/10/2016
1.3 Gestión económica y sostenible	5d	04/10/2016	10/10/2016
1.4 Presentación preliminar	5d	11/10/2016	17/10/2016
1.5 Presentación oral y entrega final	3d	13/10/2016	17/10/2016
<b>2 Recopilación de información</b>	5d	18/10/2016	24/10/2016
2.1 Búsqueda de información	3d	18/10/2016	20/10/2016
2.2 Aprendizaje autónomo	2d	21/10/2016	24/10/2016
<b>3 Implementación laboratorio virtual</b>	28d	25/10/2016	01/12/2016
<b>3.1 Instalación software</b>	7d	25/10/2016	02/11/2016
3.1.1 Análisis de requisitos	2d	25/10/2016	26/10/2016
3.1.2 Instalación y configuración	3d	27/10/2016	31/10/2016
3.1.3 Testeo	2d	01/11/2016	02/11/2016
<b>3.2 Configuración disp. móvil</b>	7d	03/11/2016	11/11/2016
3.2.1 Análisis de requisitos	2d	03/11/2016	04/11/2016
3.2.2 Configuración	3d	07/11/2016	09/11/2016
3.2.3 Testeo	1d	11/11/2016	11/11/2016
<b>3.3 Compilación cod. fuente Android</b>	5d	14/11/2016	18/11/2016
3.3.1 Análisis de requisitos	1d	14/11/2016	14/11/2016
3.3.2 Descarga, config. y compilación	2d	16/11/2016	17/11/2016
3.3.3 Testeo	1d	18/11/2016	18/11/2016
<b>3.4 Empaquetamiento herramientas y creación lab. virtual</b>	9d	21/11/2016	01/12/2016
3.4.1 Análisis de requisitos	1d	21/11/2016	21/11/2016
3.4.2 Empaquetamiento y creación lab.	5d	22/11/2016	28/11/2016
3.4.3 Testeo	3d	29/11/2016	01/12/2016
<b>4 Elaboración prácticas laboratorio</b>	20d	02/12/2016	29/12/2016
4.1 Práctica 1	4d	02/12/2016	07/12/2016
4.2 Práctica 2	4d	08/12/2016	13/12/2016
4.3 Práctica 3	4d	14/12/2016	19/12/2016
4.4 Práctica 4	4d	20/12/2016	23/12/2016
4.5 Práctica 5	4d	26/12/2016	29/12/2016
<b>5 Documentación y presentación</b>	82d	19/09/2016	10/01/2017
5.1 Documentación memoria	76d	19/09/2016	02/01/2017
5.2 Revisión final	3d	03/01/2017	05/01/2017
5.3 Presentación oral final	3d	06/01/2017	10/01/2017

Figura 4.1: Planificación de las tareas y su riesgo

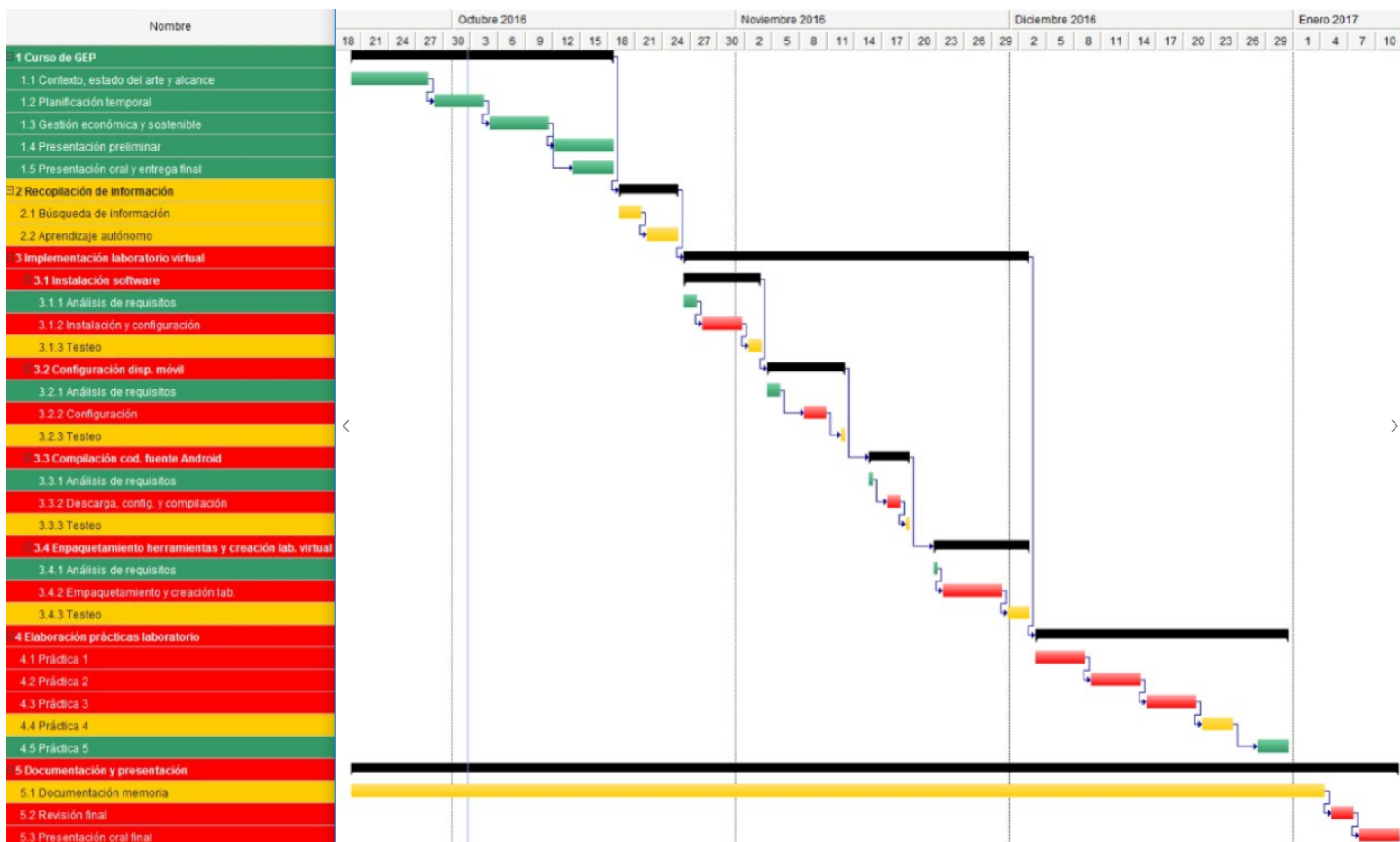


Figura 4.2: Diagrama de Gantt

# Capítulo 5

## Gestión económica

### 5.1. Consideraciones generales

Una vez diseñadas y planteadas todas las fases de las cuales se compone el proyecto, es necesario realizar un análisis de los costes que supone su realización. Estos costes, una vez calculados, determinarán si el proyecto en sí es económicamente viable o no. El cálculo inicial consiste en calcular el coste de todos los recursos, ya sean humanos, de hardware y de software. A continuación se calculan los costes de realizar cada una de las actividades del proyecto. Una vez obtenidas las cifras, se realiza un presupuesto aproximado el cuál determinará si el proyecto es o no económicamente sostenible.

### 5.2. Identificación y estimación de los costes

En el capítulo 4 se enumeran todos los recursos necesarios y se clasifican en: humanos, de hardware y de software. Para el cálculo de la remuneración que debe percibir cada rol del proyecto, se ha consultado el informe *Page Personnel* de este año 2016 [14]. En la Tabla 5.1 puede observarse la remuneración percibida según el rol y el cálculo del coste total según el número de horas de cada uno.

#### 5.2.1. Costes directos por actividad

En las Tablas 5.2 y 5.3 puede observarse el coste directo de cada actividad, tomando como referencia el coste estimado de cada rol y el número de horas por actividad.



Rol	Horas estimadas (h)	Remuneración (euro/h)	Coste estimado (euros)
Jefe proyecto	145	13,70	1986,5
Investigador	15	12,40	186
Analista	30	13,02	390,6
Desarrollador	185	7,78	1439,3
Tester	30	9,25	277,5
Programador	45	7,78	350,1
<b>Total estimado</b>	<b>450</b>		<b>4630</b>

Tabla 5.1: Coste estimado de cada rol

### 5.2.2. Costes indirectos

Los costes indirectos son todos aquellos que no afectan directamente al proyecto pero que son implícitos a él. A continuación se enumeran los costes indirectos:

- **Recursos Hardware:** El ordenador portátil es propiedad del autor del proyecto y fue adquirido con anterioridad y su propósito no fue el de realizar únicamente este proyecto sino para uso personal, por tanto, el coste se calcula en base al porcentaje de tiempo usado respecto a la vida útil del ordenador. El coste de adquisición fue de 1000 €. La vida útil es de 5 años y el tiempo de uso para el proyecto de 4 meses. Por tanto, el coste proporcional es de 67 €. El teléfono inteligente es propiedad de la Universidad Politécnica de Cataluña y es usado con otros fines, no solo para este proyecto. El coste de uso es similar al del ordenador. Su coste es de 200 € aprox. y se considera un tiempo de vida útil de 4 años. Por cuatro meses, el coste proporcional es de 17 €. La suma de ambos es de 84 €.
- **Recursos Software:** Los recursos software también tienen un coste nulo, ya que todo el software usado es *freeware*. Los servicios en la nube usados como Gmail, Google Drive y ShareLaTeX también son gratuitos.
- **Transporte:** Dos veces por semana se realizan las reuniones con el director, en su despacho. Para el desplazamiento desde casa hasta la universidad, el autor utiliza una moto. El gasto por desplazamiento teniendo en cuenta la distancia y el precio medio de la gasolina es de 0,25 céntimos de euro por trayecto. Teniendo en cuenta que el proyecto dura cuatro meses, el precio total es de aproximadamente 5 €.
- **Impresión del proyecto:** Al finalizar la redacción de la memoria, se pretende imprimir y entregar una copia para los tres miembros del tribunal y para el director. Suponiendo que la extensión del documento es de 80 páginas, que cada página tiene un coste de 0,10 € (a color) y que el coste de una encuadernación es de 2 €, el coste total es de 40 €.

En la Tabla 5.4 pueden observarse los costes indirectos.

<b>Tarea</b>	<b>Responsable</b>	<b>Horas</b>	<b>Coste</b>
<b>Curso de GEP</b>		<b>75</b>	<b>1027,5</b>
Contexto, estado del arte y alcance	Jefe proyecto	24	328,8
Planificación temporal	Jefe proyecto	10	137
Gestión económica y sostenible	Jefe proyecto	10	137
Presentación preliminar	Jefe proyecto	7	95,9
Presentación oral y entrega final	Jefe proyecto	24	328,8
<b>Recopilación Información</b>		<b>30</b>	<b>391,5</b>
Búsqueda información	Jefe proyecto	15	205,5
Aprendizaje autónomo	Investigador	15	186
<b>Implementación lab. virtual</b>		<b>150</b>	<b>1368,3</b>
<b>Subtarea 1: Instalación del software</b>		<b>40</b>	<b>378,3</b>
Análisis de requisitos	Analista	10	130,2
Instalación y configuración	Desarrollador	20	155,6
Testeo	Tester	10	92,5
<b>Subtarea 2: Configuración del dispositivo móvil</b>		<b>20</b>	<b>189,15</b>
Análisis de requisitos	Analista	5	65,1
Configuración	Desarrollador	10	77,8
Testeo	Tester	5	46,25
<b>Subtarea 3: Compilación del código fuente Android y testeo en el dispositivo móvil</b>		<b>20</b>	<b>189,15</b>
Análisis de requisitos	Analista	5	65,1
Descarga, config. y compilación	Desarrollador	10	77,8
Testeo	Tester	5	46,25
<b>Subtarea 4: empaquetamiento de todas las herramientas y creación del lab. virtual</b>		<b>70</b>	<b>611,7</b>
Análisis de requisitos	Analista	10	130,2
Empaquetamiento y creación lab.	Desarrollador 80 %	50	311,2
	Programador 20 %		77,8
Testeo	Tester	10	92,5

Tabla 5.2: Costes directos por actividad (1/2)

<b>Elaboración prácticas laboratorio</b>		<b>140</b>	<b>1089,2</b>
Elaboración práctica 1	Desarrollador 75 %	28	163,38
	Programador 25 %		54,46
Elaboración práctica 2	Desarrollador 75 %	28	163,38
	Programador 25 %		54,46
Elaboración práctica 3	Desarrollador 75 %	28	163,38
	Programador 25 %		54,46
Elaboración práctica 4	Desarrollador 75 %	28	163,38
	Programador 25 %		54,46
Elaboración práctica 5	Desarrollador 75 %	28	163,38
	Programador 25 %		54,46
<b>Documentación y presentación</b>		<b>55</b>	<b>753,5</b>
Documentación memoria	Jefe proyecto	40	548
Revisión final	Jefe proyecto	5	68,5
Presentación oral final	Jefe proyecto	10	137
<b>Total</b>		<b>450</b>	<b>4630</b>

Tabla 5.3: Costes directos por actividad (2/2)

Coste	Unidades	Precio (euros) / Unidad	Coste estimado (euros)
Hardware	1 ordenador	67	67
	1 teléfono móvil	17	17
Transporte	20 viajes	0,25	5
Impresión	320 páginas	0,10	32
	4 encuadernaciones	2	8
<b>Total</b>			<b>129</b>

Tabla 5.4: Costes indirectos

Tipo de coste	Precio (euros)	Porcentaje	Coste (euros)
directo	4630	13 %	601,90
indirecto	129	13 %	16,87
<b>Total</b>	<b>4759</b>		<b>618,67</b>

Tabla 5.5: Contingencia

### 5.2.3. Contingencia

Como plan de contingencia, se ha reservado un 13 por ciento del presupuesto, tomando como referencia los costes directos e indirectos. En la Tabla 5.5 se puede observar el resultado de aplicar el porcentaje de contingencia para ambos costes.

### 5.2.4. Imprevistos

Existe una probabilidad que el plan previsto no surja según lo planeado y que hayan imprevistos. Éstos pueden ser:

- **Avería del ordenador:** El autor del proyecto usa su propio ordenador personal. En este caso, existe una probabilidad que durante el tiempo que dura el proyecto, éste se estropee y deje de funcionar. Como el ordenador fue adquirido hace tan solo 5 meses, la probabilidad que se estropee es del 2 por ciento aproximadamente. El coste de adquirir uno nuevo de gama media sería de 560 €.
- **Avería del teléfono móvil:** Podría suceder que durante la realización del proyecto, el teléfono móvil dejara de funcionar y en ese caso habría que adquirir un terminal nuevo. Para ello, el jefe del proyecto dispone de otro dispositivo móvil de backup, por tanto, no supondría ningún coste adicional. La probabilidad que se estropee es del 5 por ciento.
- **Retraso de 2 semanas:** En el apartado 4.1.3 se hace referencia a la holgura temporal de 2 semanas, en caso de retraso en la entrega. Se considera una probabilidad de que esto suceda del 20 por ciento. El rol que ejecutaría esta tarea sería el de jefe de proyecto y supondrían 75 horas empleadas aproximadamente.

En la Tabla 5.6 se desglosa el coste de cada imprevisto según su probabilidad.

### 5.2.5. Presupuesto

El presupuesto final del proyecto tiene en cuenta estas consideraciones:

Imprevisto	Unidades	Probabilidad	Precio (euros)	Coste (euros)
Avería ordenador	1	2 %	560	11,20
Avería teléfono móvil	1	5 %	0	0
Retraso 2 semanas	75 h	20 %	1027,50	205,50
<b>Total</b>			<b>1587,50</b>	<b>216,70</b>

Tabla 5.6: Coste de los imprevistos

Concepto	Coste (euros)
Costes directos	4630
Costes indirectos	129
Contingencia	618,67
Imprevistos	216,70
<b>Total</b>	<b>5594,37</b>

Tabla 5.7: Presupuesto

- Se supone que todos los costes tienen un precio fijo. Asimismo, la remuneración de cada rol tampoco es variable a lo largo del tiempo.
- Al ser un proyecto sin ánimo de lucro, no se obtendrá ningún beneficio económico del mismo.

En la Tabla 5.7 puede observarse el presupuesto final del proyecto.

### 5.3. Control de gestión

Los recursos de software y hardware forman parte de la inversión inicial y aunque para este proyecto su coste es nulo, tampoco sería posible hacer un control constante y solo podría compararse el presupuesto con el coste final, ya que solo se invierte en ellos al inicio del proyecto.

Para los recursos humanos, sí que se puede hacer un control constante. Al finalizar cada tarea, se revisa que el número de horas trabajadas coincida con el calculado inicialmente. En caso que la diferencia sea muy notable, tanto por hacer más horas de las previstas como menos, se buscará el motivo de dicha diferencia y procurará corregirlo para las siguientes fases del proyecto, de manera que el presupuesto quede mejor ajustado a la realidad. En el caso que al sumar el coste real de todas las tareas éste sea menor que el presupuesto inicial, deberá aplicarse el plan de contingencia para subsanar esta diferencia de gastos. La manera de calcular estas desviaciones es la siguiente:

- 
- Desvío de mano de obra en precio =  $(\text{coste std} - \text{coste real}) * \text{consumo de horas real}$ .
  - Desvío en la realización de una tarea en precio =  $(\text{coste std} - \text{coste real}) * \text{consumo horas reales}$ .
  - Desvío de un recurso en precio =  $(\text{coste std} - \text{coste real}) * \text{consumo real}$ .
  - Desvío en la realización de una tarea en consumo =  $(\text{consumo std} - \text{consumo real}) * \text{coste real}$ .
  - Desvío total en la realización de tareas =  $\text{coste total std tarea} - \text{coste total real tarea}$ .
  - Desvío total en recursos =  $\text{coste total std recursos} - \text{coste total real recursos}$ .
  - Desvío total costes fijos =  $\text{coste total coste fijos presupuestado} - \text{coste total fijo real}$ .

# Capítulo 6

## Background de Android

### 6.1. Características principales

Android es un sistema operativo diseñado para ser usado en un dispositivo móvil. Inicialmente su uso era para teléfonos inteligentes. Actualmente, es también usado en tabletas, ordenadores portátiles, *netbooks*, Google TV, relojes de pulsera, y otros dispositivos [15]. Algunas de las características que lo definen como un moderno sistema operativo móvil son:



Figura 6.1: Logo oficial de Android

- **Ecosistema basado en aplicaciones:** Permite añadir y eliminar fácilmente aplicaciones y publicar nuevas características en el sistema.
- **Soporte web:** Soporta las últimas tecnologías web, con un navegador web que utiliza el motor de renderizado *Blink* [16].
- **Aceleración por hardware:** Dispone de aceleración gráfica por hardware mediante *OpenGL ES* [17], la versión de sistemas embebidos.
- **Comunicación inalámbrica:** Soporta tecnologías de comunicación inalámbrica como *GSM*, *CDMA*, *UMTS*, *LTE*, *Bluetooth*, *WiFi*, *NFC*.

## 6.2. Android Open Source Project

El *Android Open Source Project* es la plataforma oficial donde el equipo de Google publica el código fuente de Android [18]. De esta manera, cualquier desarrollador puede trabajar con el código fuente y realizar modificaciones, aunque la versión final que se publica mundialmente siempre es la desarrollada por Google. Cabe destacar que solo los dispositivos de Google (Nexus y Píxel) son soportados por AOSP. Para los demás dispositivos, se deben realizar una serie de modificaciones específicas de cada terminal.

En la propia web de AOSP están especificadas todas las versiones de Android con su versión de API (ver Figura 6.2). Además, para cada rama del proyecto se indica su nombre, la versión y los dispositivos que la soportan.

Code name	Version	API level
Nougat	7.1	API level 25
Nougat	7.0	API level 24
Marshmallow	6.0	API level 23
Lollipop	5.1	API level 22
Lollipop	5.0	API level 21
KitKat	4.4 - 4.4.4	API level 19

Figura 6.2: Nombre, versión y API de Android desde la versión 4.4 a la actual 7.1.

## 6.3. Diferencias entre el kernel de Android y Linux

En esta sección se detallan las diferencias principales de ambos kernels [19]:

- **Wakelock API:** En un dispositivo Android, para poder ahorrar batería, el sistema debe pasar a estado de suspensión cada vez que sea posible. Por ello, las aplicaciones que deban permanecer corriendo (por ejemplo cuando la pantalla esté apagada), deben usar la API de los Wakelocks, con lo cuál seguirán funcionando incluso en modo de suspensión. En la Figura 6.3 es posible observar sus funciones.
- **Binder:** Se trata de un mecanismo de comunicación entre procesos. Es el driver encargado de establecer las comunicaciones que se realizan, ya sean llamadas de sistema, comunicación entre aplicaciones e incluso entre componentes de una misma aplicación. Es la pieza clave para que Android funcione. En la Figura 6.4 puede observarse los tres tipos de comunicación.



```

#include <linux/wakelock.h>
void wake_lock_init(struct wakelock *lock,
                   int type,
                   const char *name);
void wake_lock(struct wakelock *lock);
void wake_unlock(struct wakelock *lock);
void wake_lock_timeout(struct wakelock *lock, long timeout);
void wake_lock_destroy(struct wakelock *lock);

```

Figura 6.3: Funciones de la API de Wakelocks en el espacio del kernel.

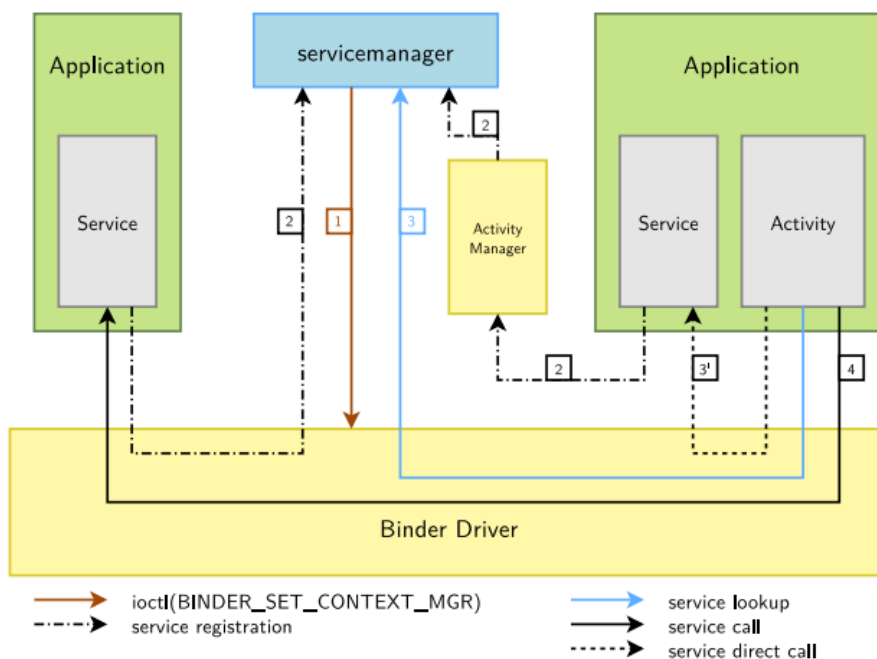


Figura 6.4: Esquema con algunas de las comunicaciones disponibles en Binder.

- **Klogger:** En el kernel de Linux todos los logs se almacenan mediante una llamada de sistema a `syslog()` a través de un socket. Esto provoca que se tenga que realizar un cambio de contexto lo que supone muchos tiempos de ciclo. Además, cada una de estas llamadas escribe en un fichero, el cual está almacenando en un dispositivo que hace que la escritura consuma también muchos tiempos de ciclo. Para subsanar estos dos problemas, se usa `klogger`, un driver del kernel que utiliza cuatro buffers en el área de memoria del kernel. Éstos se almacenan en `/dev/log` y son accesibles mediante la librería `liblog`. Esta librería es a su vez usada por el sistema y las aplicaciones para escribir los logs.
- **Ashmem (Anonymous Shared Memory):** Se trata de un mecanismo estándar de compartición de memoria entre procesos. En el kernel de Android, se añade un

contador de referencias de manera que el kernel puede descartar aquellas que ya no están en uso. También dispone de un mecanismo de contraer determinadas regiones de memoria cuando la memoria se encuentra en niveles elevados de ocupación.

- **Alarm timers:** Los relojes de alta resolución en Linux no pueden ser usados cuando el sistema está en suspensión. El reloj de tiempo real sí que puede levantar el sistema de un estado de suspensión, pero no hacerlo solo para un proceso en concreto. Teniendo en cuenta la política de ahorro de energía buscada en Android, es necesario un reloj capaz de levantar procesos cuando se está en suspensión. Los relojes alarma son la solución. Una vez el proceso/aplicación es levantado por la alarma, se lanza el evento del wakelock.
- **Low Memory Killer:** En Linux, cuando la memoria está prácticamente llena, se lanza al proceso *Out Of Memory Killer (OOM Killer)* el cual mata procesos para liberar memoria. Pero en Android, esto no es óptimo, ya que hay procesos esenciales para el correcto funcionamiento del dispositivo que el propio OOM Killer no es capaz de distinguir entre procesos clave del sistema como la pila del teléfono, el sistema gráfico, etc. con otros mucho menos prioritarios como las aplicaciones de usuario que actualmente estén abiertas. Por ello, antes de lanzar el OOM Killer, en Android se llama al *Low Memory Killer*. Para detectar qué procesos liberar, calcula el tiempo desde que la aplicación fue usada por última vez junto con su prioridad. La idea es que cuando se aplica este *killer*, el OOM Killer no será nunca llamado ya que se habrá conseguido liberar memoria suficiente.
- **ION Memory Allocator:** En Linux se permiten alocar regiones de memoria de 512 páginas (con tamaños de página de 4KB). Para Android es interesante que este tamaño sea mucho más grande. Por ejemplo, cuando se obtiene una imagen de la cámara y se quiere subir a un codificador JPEG el cual es otra aplicación de usuario. Dispone de una API de usuario para alocar memoria entre aplicaciones.
- **Seguridad de red:** Linux permite abrir sockets fácilmente y establecer una comunicación con el exterior. Pero Google deseaba restringir estas conexiones aplicando permisos en cada aplicación. Estos permisos son necesarios si alguna aplicación desea comunicarse vía IP, Bluetooth, sockets raw (sin ningún protocolo de transporte determinado) o RFCOMM (comunicaciones serie vía Bluetooth).

## 6.4. Proceso de boot

Android, al ser un sistema operativo como tal, se inicia de forma similar al resto de sistemas operativos. Tiene, no obstante, algunas diferencias respecto al boot de un sistema basado en Linux. Los pasos de boot en Android son [20]:

1. **Encendido del terminal y ejecución del código ROM:** Se inicia el código de boot de la unidad ROM. Se trata de un código con una dirección inicial fija y conocida. Éste se encarga de localizar y ejecutar el *boot loader* en la memoria RAM.
2. **Boot loader:** Es un programa separado del kernel de Linux encargado de inicializar la memoria y cargar el kernel en la RAM. Al finalizar la carga, se continua la ejecución con el kernel ya en RAM.
3. **Kernel de Linux:** El proceso es muy similar al de cualquier otro sistema Linux de escritorio. Entre otros, se encarga de inicializar los controladores de interrupción, las protecciones de memoria, cachés y los *schedulers* . Una vez el sistema ha sido cargado, ya es posible de acceder al proceso *init* del sistema de ficheros (localizado en `/system/core/init`) e iniciarlo como un proceso del espacio de usuario.
4. **Proceso init:** Es el primer proceso que se ejecuta. Todos los siguientes procesos serán hijos de este. En el caso de Android, éste se encarga de localizar el fichero *init.rc* (localizado en `/system/core/rootdir`). Este script contiene la descripción de los servicios del sistema, , sistema de ficheros y otros parámetros que se encarga de configurar. Al finalizar, llama a los procesos del servicio de sistema.
5. **Zygote y Dalvik:** Zygote es el servicio de sistema encargado de lanzar la máquina virtual Dalvik. Básicamente su función es crear máquinas virtuales por cada proceso nuevo que se inicia.
6. **System Server:** Es el primer componente de Java que se ejecuta en el sistema. Inicializa todos los servicios de Android como el gestor del teléfono, el módulo Bluetooth, etc.
7. **Finalización del boot:** En este último paso, se envía un mensaje broadcast ACTION\_BOOT\_COMPLETED el cual es posible capturar si se desea iniciar un servicio o aplicación nada más terminar el proceso de boot. A partir de este momento, el boot ha finalizado.

## 6.5. Particiones

Android, al igual que Linux, permite estructurar el sistema de ficheros en una serie de particiones cada una con un propósito distinto. Éstas (exceptuado la `sdcard`), se almacenan en una memoria interna de estado sólido (flash) llamada NAND. Las particiones comunes a todo dispositivo Android son [21]:

- **/boot** : Necesaria para el arranque del sistema. Dentro se incluye el *boot loader* y el kernel de linux. El dispositivo arranca el sistema por defecto en esta partición.
- **/recovery** : Representa el arranque alternativo. Permite realizar una recuperación del sistema, así como borrar datos del dispositivo, instalar una actualización del sistema o incluso una nueva ROM si el terminal ha sido rooteado. En el caso de disponer de un recovery distinto al original como *TWRP 3.0-2-0* se disponen de más funciones como: backups, restores, mounts, modo consola, particionado de la tarjeta SD, etc. En la Figura 6.5 se ven algunas capturas de este recovery.

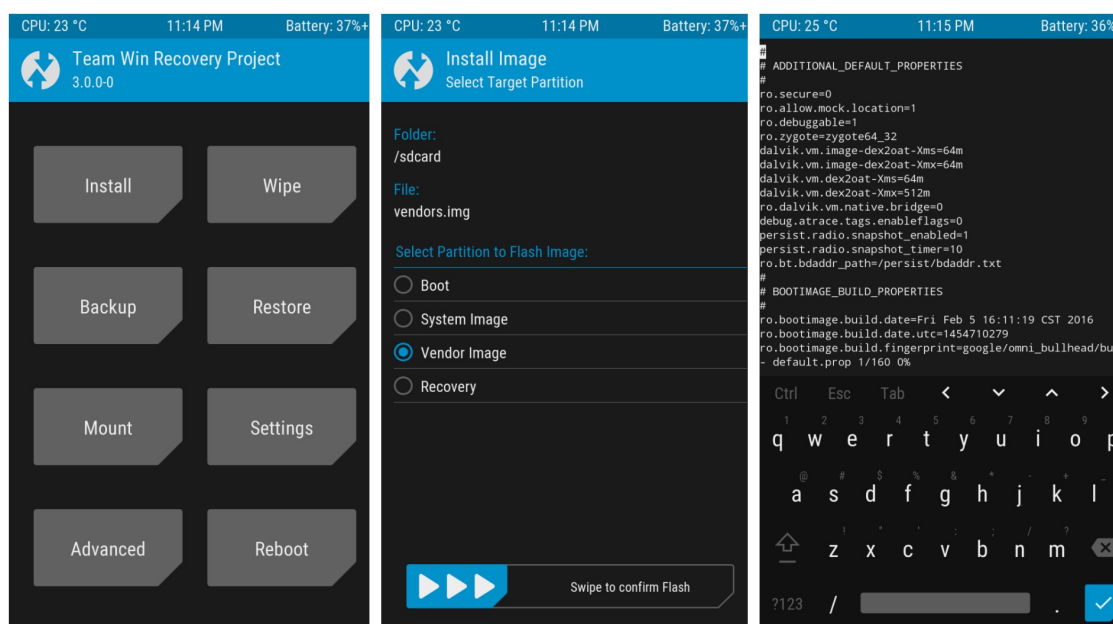


Figura 6.5: Menú principal, modo instalación y modo shell del Recovery TWRP 3.0-2-0.

- **/system** : Contiene el sistema operativo y las aplicaciones.
- **/cache** : Los datos caché se almacenan en esta partición. Su borrado no compromete la integridad del sistema, ya que estos datos se pueden regenerar por las propias aplicaciones.
- **/misc** : Contiene datos vitales necesarios para la correcta configuración del terminal, como parámetros del operador de red, configuración del USB, etc. No se recomienda modificarla.
- **/data** : Almacena toda la información del usuario, ya sea los contactos, datos de aplicaciones, contenido multimedia, etc. Borrar el contenido de esta partición supone dejar el terminal en modo de fábrica.

- **/sdcard** : Esta partición existe en el caso de tener instalada en el terminal una tarjeta SD interna. Es usada como ampliación de la partición /data. En ella se suele almacenar contenido multimedia. Algunas aplicaciones permiten guardar sus datos en esta partición (así se libera espacio en /data).

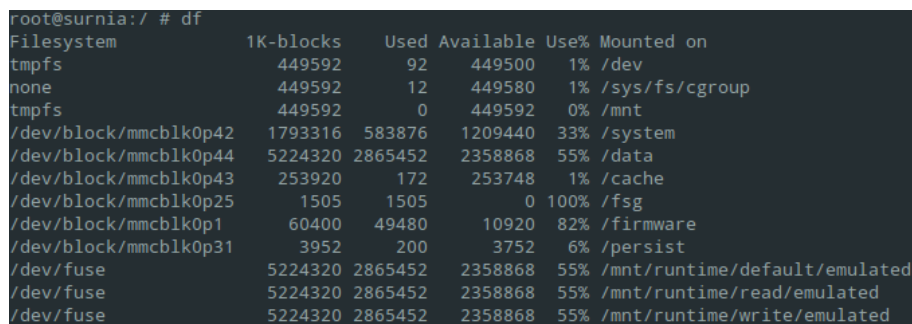
Cada dispositivo contiene además sus propias particiones. Una muy común es **/radio** la cual contiene el *firmware* de la radio, controla la red móvil, los datos móviles, el GPS y el Bluetooth. **/sd-ext** es usada para tarjetas SD externas. Muchas ROMs permiten que esta partición tenga el mismo uso que la /data, para ampliar el espacio disponible de datos de usuario.

Para visualizar las particiones montadas en el dispositivo usado en este proyecto (Motorola LTE E), se ha entrado a la *shell* del terminal y lanzado el siguiente comando (el resultado puede observarse en la Figura 6.6):

```
$ df
```

Pero para visualizar la totalidad de particiones existentes en el sistema, debe ejecutarse el siguiente comando (el resultado se observa en la Figura 6.7):

```
$ ls -l /dev/block/platform/msm_sdcc.1/by-name/
```



```

root@surnia:/ # df
Filesystem            1K-blocks    Used Available Use% Mounted on
tmpfs                  449592         92  449500    1% /dev
none                   449592         12  449580    1% /sys/fs/cgroup
tmpfs                  449592          0  449592    0% /mnt
/dev/block/mmcblk0p42 1793316   583876 1209440   33% /system
/dev/block/mmcblk0p44 5224320 2865452 2358868   55% /data
/dev/block/mmcblk0p43 253920     172  253748    1% /cache
/dev/block/mmcblk0p25 1505       1505      0 100% /fsg
/dev/block/mmcblk0p1 60400     49480  10920   82% /firmware
/dev/block/mmcblk0p31 3952       200   3752    6% /persist
/dev/fuse              5224320 2865452 2358868   55% /mnt/runtime/default/emulated
/dev/fuse              5224320 2865452 2358868   55% /mnt/runtime/read/emulated
/dev/fuse              5224320 2865452 2358868   55% /mnt/runtime/write/emulated

```

Figura 6.6: Particiones montadas en el dispositivo Motorola LTE E.

## 6.6. Sistema de ficheros

Al igual que en Linux, Android es capaz de acceder a diversos formatos de sistemas de ficheros. El más común y usado (al igual que en entornos Linux de escritorio) es el *ext4*, debido a sus características como su capacidad de realizar *journaling* [23], su mejorado

```

root@surnia:~# ls -l /dev/block/platform/msm_sdcc.1/by-name/
total 0
lrwxrwxrwx 1 root root 20 1970-06-16 09:47 DDR -> /dev/block/mmcblk0p3
lrwxrwxrwx 1 root root 20 1970-06-16 09:47 aboot -> /dev/block/mmcblk0p4
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 abootBackup -> /dev/block/mmcblk0p14
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 boot -> /dev/block/mmcblk0p33
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 cache -> /dev/block/mmcblk0p43
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 carrier -> /dev/block/mmcblk0p41
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 cid -> /dev/block/mmcblk0p28
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 clogo -> /dev/block/mmcblk0p30
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 dhob -> /dev/block/mmcblk0p24
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 factorytune1 -> /dev/block/mmcblk0p11
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 factorytune2 -> /dev/block/mmcblk0p35
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 frp -> /dev/block/mmcblk0p20
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 fsc -> /dev/block/mmcblk0p26
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 fsg -> /dev/block/mmcblk0p25
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 hob -> /dev/block/mmcblk0p23
lrwxrwxrwx 1 root root 20 1970-06-16 09:47 hyp -> /dev/block/mmcblk0p7
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 hypBackup -> /dev/block/mmcblk0p18
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 keystore -> /dev/block/mmcblk0p39
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 kpan -> /dev/block/mmcblk0p36
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 logo -> /dev/block/mmcblk0p29
lrwxrwxrwx 1 root root 20 1970-06-16 09:47 logs -> /dev/block/mmcblk0p9
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 metadata -> /dev/block/mmcblk0p13
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 misc -> /dev/block/mmcblk0p32
lrwxrwxrwx 1 root root 20 1970-06-16 09:47 modem -> /dev/block/mmcblk0p1
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 modemst1 -> /dev/block/mmcblk0p21
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 modemst2 -> /dev/block/mmcblk0p22
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 oem -> /dev/block/mmcblk0p40
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 padA -> /dev/block/mmcblk0p12
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 padB -> /dev/block/mmcblk0p19
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 padC -> /dev/block/mmcblk0p37
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 persist -> /dev/block/mmcblk0p31
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 recovery -> /dev/block/mmcblk0p34
lrwxrwxrwx 1 root root 20 1970-06-16 09:47 rpm -> /dev/block/mmcblk0p5
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 rpmBackup -> /dev/block/mmcblk0p15
lrwxrwxrwx 1 root root 20 1970-06-16 09:47 sbl1 -> /dev/block/mmcblk0p2
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 sec -> /dev/block/mmcblk0p10
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 sp -> /dev/block/mmcblk0p38
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 ssd -> /dev/block/mmcblk0p27
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 system -> /dev/block/mmcblk0p42
lrwxrwxrwx 1 root root 20 1970-06-16 09:47 tz -> /dev/block/mmcblk0p6
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 tzBackup -> /dev/block/mmcblk0p16
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 userdata -> /dev/block/mmcblk0p44
lrwxrwxrwx 1 root root 20 1970-06-16 09:47 utags -> /dev/block/mmcblk0p8
lrwxrwxrwx 1 root root 21 1970-06-16 09:47 utagsBackup -> /dev/block/mmcblk0p17

```

Figura 6.7: Particiones existentes en el dispositivo Motorola LTE E.

diseño respecto ext2 y ext3, soporte a mayor tamaño de ficheros, menor consumo de CPU, etc [24].

No obstante, aprovechando que el sistema está montado sobre una tarjeta flash, como alternativa se puede optar por usar *F2FS (Flash-Friendly File System)*, un sistema de ficheros creado por Samsung y añadido al kernel de Linux 3.6 en 2012 [25]. Este sistema está optimizado para ser usado en memorias tipo flash. Una de las prácticas de laboratorio propuestas en este proyecto consiste en testear y comparar estos dos sistemas de ficheros.

# Capítulo 7

## Laboratorio virtual

### 7.1. Consideraciones generales

Como se especificó en el apartado 3.1, el laboratorio virtual reúne todas las herramientas necesarias para poder trabajar con el kernel de Android. En este capítulo se hablará de los requerimientos para poder hacerlo funcionar, sus distintos componentes, así como conceptos básicos que se deben conocer para trabajar con el entorno. Finalmente, se sugieren tres métodos de implementación final.

### 7.2. Análisis de requisitos

El laboratorio virtual está compuesto por distintos paquetes de software, los cuales deben tener la versión mínima requerida para funcionar bien conjuntamente. Además, también requiere de un hardware con unas características concretas para poder funcionar. Estos datos están extraídos de la web oficial de *Android Open Source Project (AOSP)* [26] y de la web de *CyanogenMod* [27]. A continuación, se especifican estos requerimientos de hardware y de software.

#### 7.2.1. De hardware

En la Tabla 7.1 se definen las características mínimas y recomendadas de hardware.

Característica	Mínimo	Recomendado	Comentarios
<b>Ordenador</b>			
Arquitectura procesador	64 bits	64 bits	Para versiones de Android inferiores a la 2.3 es posible usar arquitectura de 32 bits.
Espacio disponible disco duro	100 GB	150 GB	En caso de compilar para múltiples dispositivos o usar ccache se recomiendan 150 GB.
Memoria RAM	2 GB	<8GB	Contra más RAM más rápida la compilación. En caso de usar Linux virtualizado, se recomiendan 16 GB de RAM/SWAP.
<b>Teléfono móvil</b>			
Tipo	Android	Android	Todo el proyecto está basado en Android.
<b>Otros</b>			
Cable USB			Dependiendo del dispositivo será tipo microUSB o miniUSB.

Tabla 7.1: Requerimientos hardware

### 7.2.2. De software

Dependiendo de la versión de Android que se desee compilar, las versiones de software cambian. Los requerimientos de software se han dividido en dos según el papel que representan dentro del laboratorio virtual. Por una parte la compilación del código fuente y por la otra la comunicación con el dispositivo, modificación del código fuente y desarrollo de aplicaciones en Android.

En la Tabla 7.2 se especifican los requerimientos de software para poder compilar desde cero el código fuente de Android 6.0 (Marshmallow).

Para la modificación del código fuente, así como el desarrollo de aplicaciones para Android y la comunicación con el dispositivo es necesario el software especificado en la Tabla 7.3.

## 7.3. Dispositivo físico

Para realizar las pruebas de Android, es necesario disponer de un dispositivo físico (teléfono móvil), el cual sea capaz de ejecutar Android 6.0. Para este proyecto, el director de este proyecto ha facilitado al autor un teléfono móvil *Motorola Moto E (LTE)* [28] de color blanco. Para que el dispositivo pueda ser usado para el laboratorio, deben realizarse una serie de modificaciones software, las cuales están descritas en la web de *MovilZona.es* [29]. Hay que tener en cuenta que al realizar estas modificaciones se pierde la garantía del dispositivo. Éstas son (ordenadas secuencialmente):



Android 6.0 (Marshmallow)		
Característica	Versión	Comentarios
Sistema Operativo	Ubuntu 14.04 LTS Mac OS v10.10 (Yosemite)	No es posible compilar Android en sistemas Windows.
Java Development Kit (JDK)	OpenJDK 8 (Ubuntu) jdk 8u45 (Mac OS)	La versión para Mac OS puede ser la especificada o una superior.
Python	2.6, 2.7	De pythong.org
GNU Make	3.81, 3.82	De gnu.org
Git	1.7 mínima	De git-scm.com
Librerías y paquetes	bison build-essential curl flex gnupgperf libesd0-dev liblz4-tool libncurses5-dev libstdc++6 libwxgtk2.8-dev libxml2 libxml2-utils lzop maven pngcrushschedtool squashfs-tools xsltproc zipzlib1g-dev g++-multilib gcc-multilib lib32ncurses5-dev lib32readline-gplv2-dev lib32z1-dev	Necesarios para la compilación del código fuente en arquitectura x64.

Tabla 7.2: Requerimientos software para la compilación del código fuente de Android 6.0

Android 6.0 (Marshmallow)		
Plataforma	Versión	Comentarios
Android SDK	6.0	Necesario para usar la API de Android 6.0.
Android SDK Tools	24.3	Contiene un conjunto de herramientas de desarrollo y debugging.
IDE para Android	-	Se recomienda Android Studio (con Android SDK incluido)
ADB	-	Herramienta de comunicación (host - dispositivo). Ya incluida en Android SDK.
VirtualBox	5.0.4	Versión mínima para ejecutar algunos emuladores de Android.

Tabla 7.3: Requerimientos software para el desarrollo, comunicación y depuración de aplicaciones Android 6.0

1. **Desbloqueo del bootloader:** Se trata del código encargado de ejecutar el sistema operativo después de realizar varios tests automáticos, dejando el terminal en un estado inicial conocido. Se almacena en un área de memoria segura y el código es específico de cada modelo de terminal. Desbloquearlo significa permitir escoger un sistema operativo distinto al original del fabricante.
2. **Instalación del recovery:** Es una partición independiente al sistema operativo de Android. Tiene su propio kernel y sirve para realizar actualizaciones de software, borrado de datos de usuario o ejecutar herramientas externas desde una tarjeta microSD. Este paso es necesario para más adelante instalar un sistema operativo diferente al original.
3. **Obtener acceso root:** Este paso, una vez realizado, permite tener acceso a funciones más avanzadas que con un usuario sin privilegios no podrían usarse. Por ejemplo, mover aplicaciones a la tarjeta SD, desinstalar aplicaciones preinstaladas, crear area de SWAP, etc. Para este proyecto, este paso es necesario también para poder instalar otro sistema operativo.
4. **Instalación de la ROM:** Se trata de un sistema operativo modificado basado en Android. Para este proyecto, se ha decidido instalar la ROM *CyanogenMod 12.1* por ser una ROM estable y una de las más usadas [30].

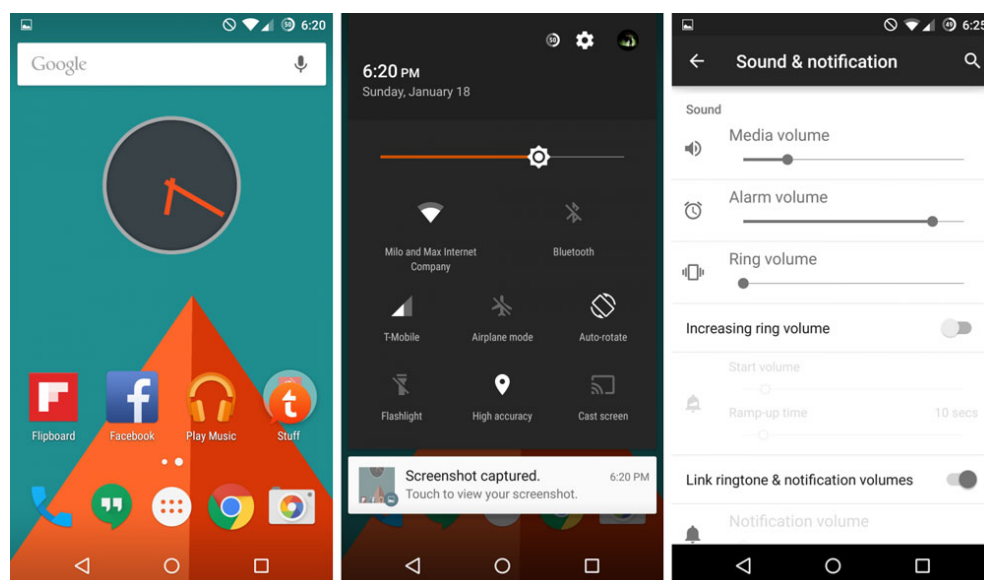


Figura 7.1: Captura de pantalla de CyanogenMod

## 7.4. Emuladores

Una pieza clave del laboratorio virtual es el emulador virtual el cual permite emular Android OS desde un sistema operativo de escritorio. Existen muchas alternativas de software de emulación para Android. El autor de este proyecto ha realizado una búsqueda de emuladores, revisando las características de cada uno y al final se sugiere el que puede servir para el laboratorio virtual. A continuación se describen los cinco emuladores distintos que se han consultado para este estudio:

- **Android Virtual Device (Android Emulator) [31]:** Se trata del emulador por defecto de Android Studio. Permite crear un perfil hardware personalizado con las características del dispositivo móvil, así como lanzar cualquier versión de Android e incluso emular una modificación de éste. Las instrucciones son traducidas a arquitectura ARM, por lo cual la velocidad de ejecución es menor a la nativa del propio ordenador. Dispone de funcionalidades adicionales como la de "núcleo múltiple" la cual se aprovecha en procesadores de multinúcleo para mejorar el rendimiento. Para instalar una nueva aplicación basta con arrastrar el fichero APK hasta la pantalla del emulador.



Figura 7.2: Captura de pantalla del Android Emulator

- **Genymotion [32]:** Es otro emulador de Android el cual, a diferencia del anterior, es capaz de mantener una arquitectura x86 a la hora de ejecutar Android y sus aplicaciones. Esto hace que el rendimiento sea superior. Permite, entre otros, modificar los

niveles de la batería, posicionamiento GPS, usar la webcam del ordenador, etc. Está disponible para Windows, Mac OS X y Linux. Además, es compatible con Android Studio. Dispone también de una herramienta de comandos propia llamada *GMTool* que permite comunicarse con el dispositivo virtual mediante comandos.

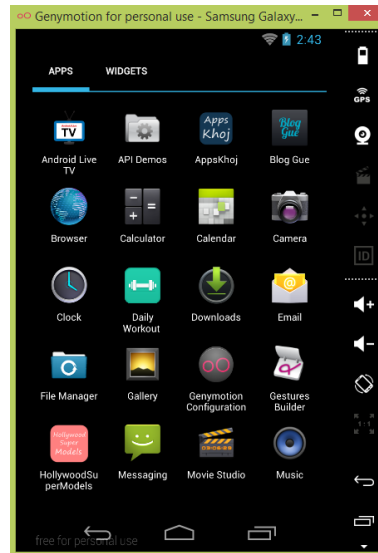


Figura 7.3: Captura de pantalla del Genymotion

- **BlueStacks [33]:** Fue creado en 2011 y su idea es la de poder ejecutar fácilmente cualquier aplicación móvil en el ordenador, aprovechándose del tamaño de la pantalla. Tiene optimizaciones de escritorio como por ejemplo el uso del teclado y el ratón para controlar el movimiento. No requiere de ningún conocimiento técnico de Android para usarse. Simplemente se escoge la aplicación de Android que se quiera instalar, y se añade como una aplicación más de escritorio. Cabe destacar que no está disponible para plataformas Linux.
- **Android-x86 [34]:** Se trata de un proyecto *opensource* con soporte para plataformas x86. Todo el código fuente está disponible en el repositorio Git de la comunidad, listo para descargar y ser compilado. Entre otros, este software ofrece soporte para la comunicación Wifi del emulador, control del estado de la batería, uso del ratón como dispositivo de control, almacenamiento externo, bluetooth, sensor de gravedad, etc. A fecha de Agosto de 2016 se está portando Android 6.0 a x86.
- **YouWave [35]:** Está pensado para el usuario que busca una manera fácil de jugar a aplicaciones Android desde el ordenador. Solo es compatible con Windows y acepta Android 4.0 (con cuenta *Premium* es posible usar la versión 5.1). Acepta almacenamiento de tarjeta SD y modo multijugador para juegos en línea. Por tanto, su uso es básicamente de entretenimiento.

Característica / Emulador	AVD	Genymotion	BlueStacks	Android-x86	YouWave
Plataformas	-Windows -Linux -Mac OS X	-Windows -Linux -Mac OS X	-Windows -Mac OS X	-Windows -Linux	Windows
Arquitectura	emulación ARM	x86	x86	x86	x86
Precio	Gratis	Gratis (disponible versión business de pago)	Gratis	Gratis	-Gratis (Android 4.0) -Pago (Android 5.1)
Licencia	Freeware	-Freeware -De pago	Freeware	Open Source (Apache 2.0)	-Freeware -De pago
Permite otro kernel?	Sí	No	No	Sí	Sí
Permite Android 6.0?	Sí	Sí	Sí	Sí	No
Compatible Android SDK?	Sí	Sí	No	Sí	No
Compatible Android Studio?	Sí	Sí	No	No	No
Compatible Eclipse?	Sí	Sí	No	No	No
Modo debug?	Sí	Sí	No	Sí	No
Comunicación dispositivo	-GUI -Comandos	-GUI -Comandos	GUI	-GUI -Comandos	GUI
Cambiar variables estado (batería, GPS, etc.)	Sí	Sí	No	Sí	No
Recomendado para	-Desarrollador -Tester	-Desarrollador -Tester	-No profesional	-Desarrollador -Tester	-No profesional

Tabla 7.4: Comparativa de las características de cada emulador

### 7.4.1. Comparativa entre emuladores

En la Tabla 7.4 se comparan las características más destacables de cada uno de los cinco emuladores.

De la tabla de comparativas se pueden diferenciar dos grupos de emuladores: los indicados para un público de desarrolladores en Android (AVD, Genymotion, Android-x86), y los de un público que busca entretenimiento y no son profesionales (BlueStacks, YouWave). Por tanto, se descartan estos dos últimos de ser usados para el laboratorio virtual, ya que no disponen de suficientes funcionalidades necesarias para el manejo de Android como se pretende.

Cada uno de los tres emuladores potencialmente viables se ejecutan de una manera diferente. En Android Emulator, cada vez que se ejecuta el emulador se crea una máquina virtual de *Qemu* [36] para emular el cambio de arquitectura. En Genymotion, se requiere del software *Oracle VM VirtualBox* [37] para ejecutarse. El mismo software se encarga de crear y gestionar la maquina virtual de manera transparente para el usuario. En el caso de Android-x86, se trata de una imagen ISO la cual puede ser instalada tanto como una nueva partición de disco como desde una máquina virtual como por ejemplo VirtualBox. En este último caso, el usuario debe definir las características hardware de la máquina virtual (en Genymotion lo realiza él mismo según el tipo de dispositivo móvil que se desee emular).

### 7.4.2. Tests de rendimiento

A continuación se muestran una serie de tests de rendimiento realizados entre los tres emuladores potencialmente viables de ser usados para el laboratorio virtual.

El primer test calcula el tiempo de arranque de Android para cada emulador. Es decir, desde que se lanza el emulador hasta que se muestra la pantalla principal de Android. Este cálculo se ha realizado mediante un cronómetro convencional. En algún casos se observa que a partir de la segunda ejecución el tiempo de arranque es mucho menor, lo cual supone una gran ventaja. La diferencia más notable es en AVD, donde disminuye 40 segundos entre ambas ejecuciones. Los resultados se muestran en la Figura 7.4.

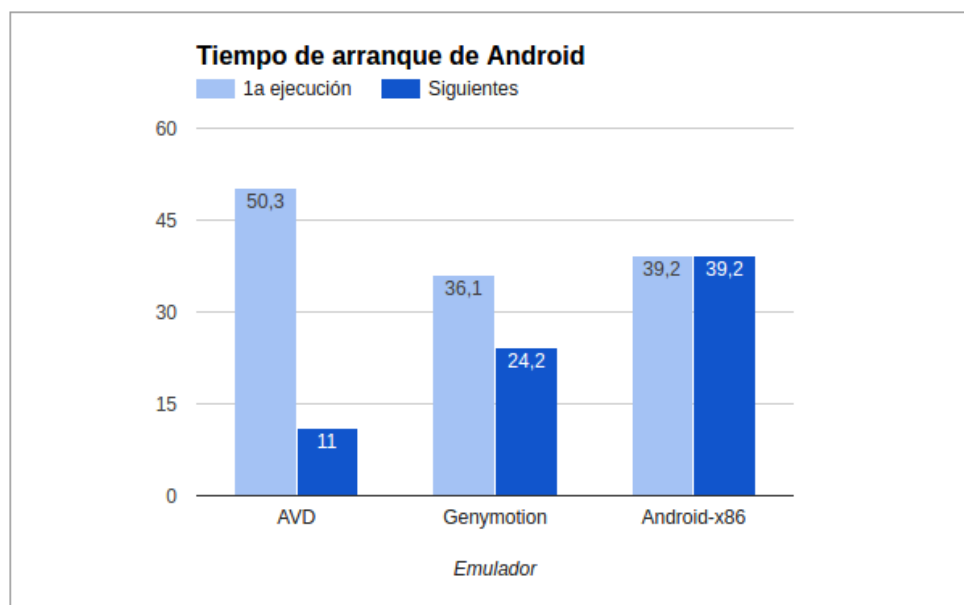


Figura 7.4: Tiempo de arranque de Android de cada emulador (segundos)

El siguiente test calcula el consumo promedio de procesador y memoria RAM de cada emulador, cuando no está ejecutando ninguna aplicación (pantalla principal de Android). Los resultados se observan en las Figuras 7.5 y 7.6. A nivel de procesador, el que más consume es ADV debido a que debe emular la arquitectura ARM y eso supone una mayor carga de trabajo para la CPU. En cuanto a los x86, el de menor consumo es Genymotion, llegando a consumir un 6,3% menos que Android-x86. En cuanto al consumo de memoria RAM, otra vez Genymotion obtiene el mejor resultado. El que más memoria ocupa es Android-x86.

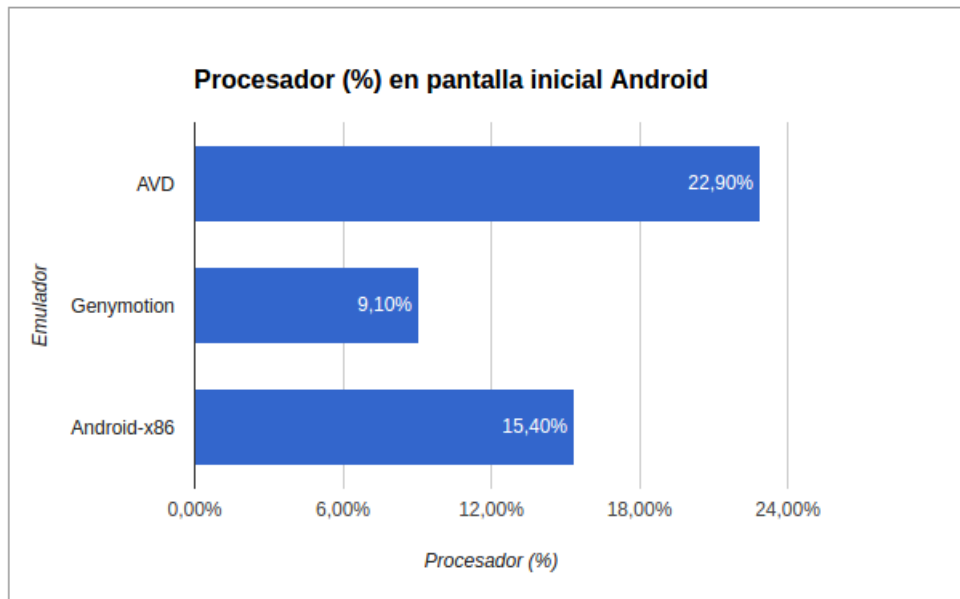


Figura 7.5: Consumo de CPU en la pantalla inicial de Android de cada emulador (%).

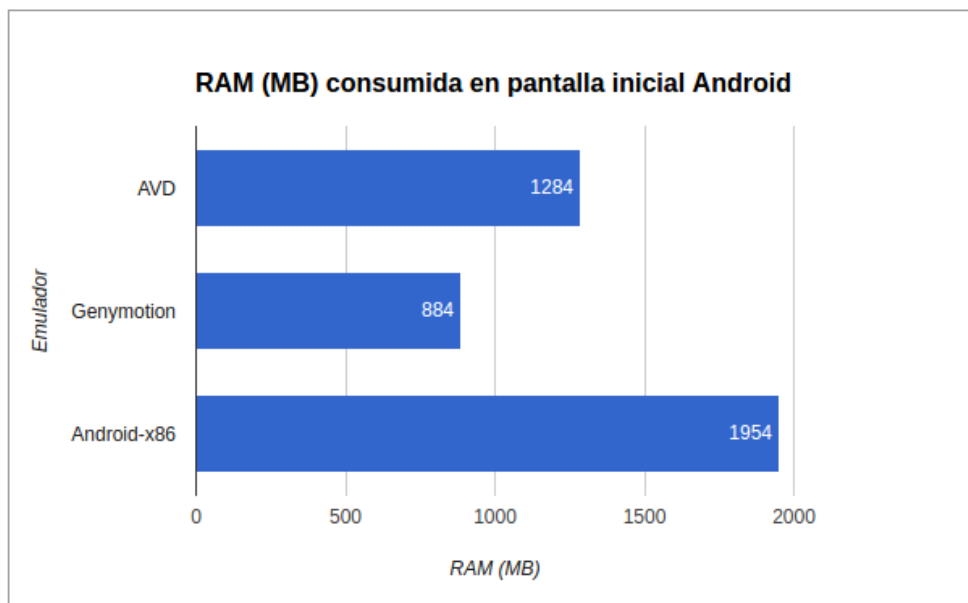


Figura 7.6: Consumo de Memoria RAM en la pantalla inicial de Android de cada emulador (MB).

El último test realizado utiliza la aplicación *AnTuTu Benchmark v6.2.1* [38] la cual calcula de manera prácticamente objetiva el rendimiento real del dispositivo Android. Entre otros aspectos, tiene en cuenta el rendimiento 3D, el rendimiento de la memoria RAM,

el rendimiento de la CPU y la experiencia de usuario (UX). Se ha instalado en los tres emuladores y el resultado obtenido se puede observar en la Figura 7.7 donde se ha dividido la puntuación en cada uno de los aspectos anteriores. En la Figura 7.8 se muestra la puntuación total. Cabe destacar que en el caso de AVD y Genymotion se obtiene una mejor puntuación gracias a que en el apartado de rendimiento de CPU destaca en el paralelismo multinúcleo durante su ejecución. Esto se ha podido observar mediante el comando *htop* durante la ejecución de AnTuTu Benchmark, en el cual se veía a todos los núcleos a niveles muy parecidos. En el caso de Android-x86 no ha sido así.

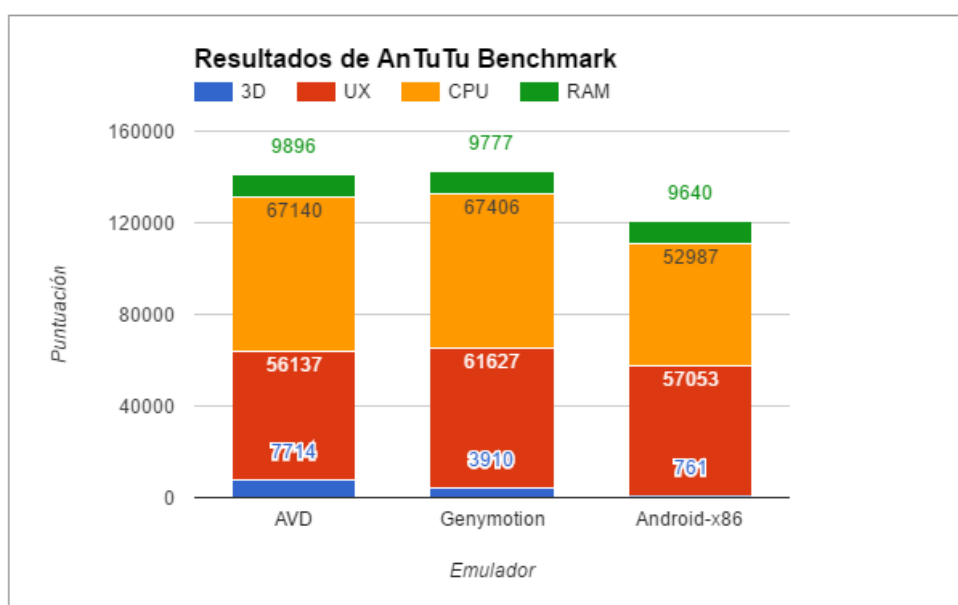


Figura 7.7: Resultados de AnTuTu Benchmark divididos en 3D, UX, CPU y RAM por emulador.

### 7.4.3. Conclusiones

Una vez analizados los aspectos técnicos de cada emulador, así como realizadas las pruebas de rendimiento, se debe decidir qué emulador incluir dentro del laboratorio virtual. Dejando de lado los emuladores dedicados al entretenimiento como BlueStacks o YouWave, existen tres emuladores candidatos: AVD, Genymotion y Android-x86.

De las características de cada emulador, se destaca que AVD es para arquitectura ARM, compatible con Android Studio y es el emulador más común a la hora de desarrollar código en Android. De Genymotion se destaca que es para arquitectura x86, es fácil de usar y su interfaz gráfica facilita el cambio de variables de estado. Por último, de Android-x86 se



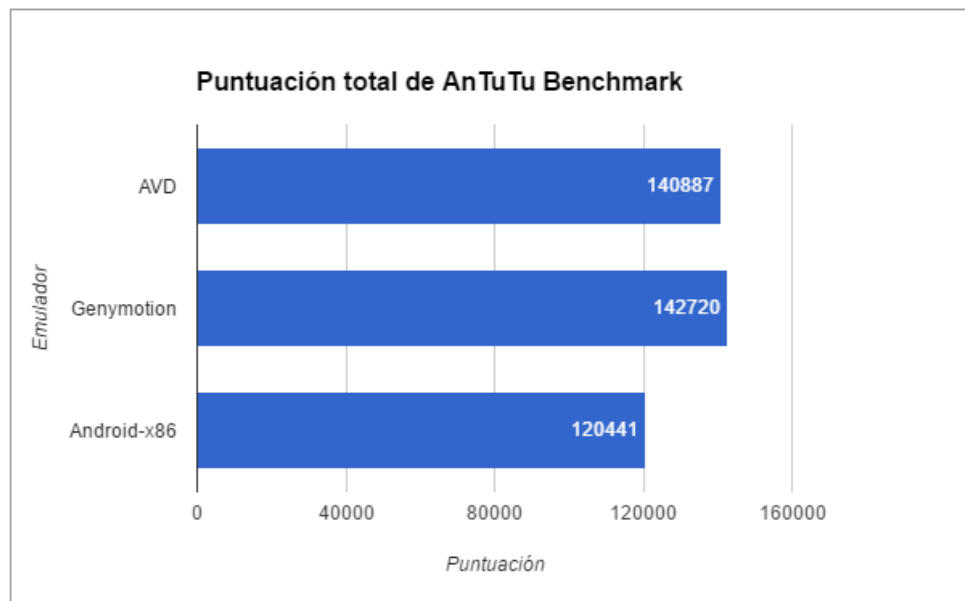


Figura 7.8: Puntuación total de AnTuTu Benchmark de cada emulador

destaca que usa arquitectura x86, es un proyecto Open Source y es posible de ser instalado como una partición de disco.

Revisando las pruebas de rendimiento, se observa que AVD y Genymotion son los que arrancan más rápido (en el caso de AVD a partir de la segunda vez) de los tres. También obtienen mejores resultados que Android-x86 en cuanto a consumo de memoria RAM. En la carga de procesador, AVD sale perjudicado, ya que utiliza una arquitectura ARM que debe emular y esto requiere más consumo de CPU. En cuanto a las pruebas de Benchmark, AVD y Genymotion siguen obteniendo mejores resultados que Android-x86.

Tras considerar las ventajas e inconvenientes de cada emulador, se concluye que deberían usarse para el laboratorio virtual AVD y Genymotion. Uno para trabajar en arquitectura ARM (misma arquitectura que un dispositivo físico) y otro para arquitectura x86. Por tanto, ambos emuladores deben formar parte del laboratorio virtual.

## 7.5. Código fuente

### 7.5.1. Descarga

El primer paso para compilar Android es obtener su código fuente. Para ello, existen varias opciones: o bien descargarlo de la fuente oficial desde la web de Android Open Source Project (AOSP) [26] o a través de una fuente no oficial, como por ejemplo la web de alguna ROM de Android. Hay que tener en cuenta que con esta segunda opción se obtiene una modificación del código fuente respecto de la versión oficial de AOSP.

El código fuente Android de AOSP es por tanto el oficial. Por ello, el desarrollador que lo utiliza se asegura que ha sido testado anteriormente antes de ser publicado y en principio no supone un riesgo para el terminal al tratarse de una fuente oficial. No obstante, esta versión una vez compilada solo es compatible con los teléfonos móviles oficiales de Android, como el modelo *Nexus* o el nuevo modelo *Pixel* que ha salido al mercado en 2016 [39]. En la web de AOSP se muestra una tabla del dispositivo móvil compatible con cada versión del código fuente [40].

La segunda opción es la descarga del código fuente de alguna ROM de Android. Teniendo en cuenta que para el laboratorio virtual se utiliza un terminal Motorola Moto E LTE, se debe encontrar una ROM compatible con este dispositivo. Después de realizar una búsqueda de ROMs compatibles, se concluye que tan solo existe una disponible: la ROM de *CyanogenMod*, en concreto la versión *surria* que es la versión compatible con el Moto E LTE [30]. En la misma web están las instrucciones de cómo preparar el dispositivo para que sea reemplazado por una nueva ROM, tal y como se resume en el apartado 7.3.

A partir de este punto, se ha tomado la decisión de usar la ROM de CyanogenMod, al ser la única compatible con el dispositivo móvil. Todo el código fuente está disponible en la página *GitHub* de CyanogenMod [41]. Ésta está dividida en repositorios, cada uno del cual representa un módulo de Android. Gracias a esta división, se consigue gestionar mejor cada módulo y dividir mejor la construcción de cada uno. Para inicializar el proceso de descarga del código fuente, es necesario instalar previamente el comando *repo* el cual, entre otros, permite descargar todos los repositorios e unirlos una vez están en local. Esta herramienta fue desarrollada por Google para poder gestionar todos los repositorios Git de un mismo proyecto de Android así como también automatizar el proceso de desarrollo [42].

Una vez se ha instalado *repo*, se debe inicializar con la versión de CyanogenMod que se desee descargar y finalmente ejecutar el comando encargado de sincronizar todos los repositorios:

```
$ repo sync
```

En el caso de CyanogenMod, el tamaño aproximado en disco de todo el código fuente una vez descargado es de 14,2 GB como se observa en la Figura 7.9. El tiempo de descarga depende de la velocidad de conexión. El autor realizó la descarga a través de línea de fibra óptica de 100Mbs y el tiempo total fue de 2 horas 15 minutos desde que se ejecutó el comando hasta que finalizó. En las siguientes sincronizaciones, el tiempo es mucho menor ya que solo descarga los cambios realizados desde la anterior sincronización.

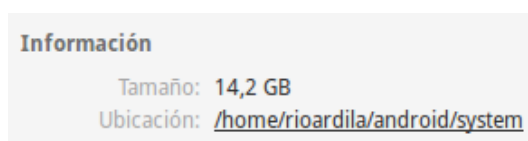


Figura 7.9: Tamaño que ocupa en disco el código fuente de CyanogenMod.

### 7.5.2. Compilación

Una vez descargado todo el código fuente, hay que preparar el entorno de trabajo para que sea capaz de compilarlo, por ejemplo, definiendo alias o variables globales. La ejecución del siguiente script se encarga de realizar esta tarea:

```
$ source build/envsetup.sh
```

A continuación, es necesario especificar el modelo o *target* del dispositivo para el cuál va a ser usado. En el caso del Motorola LTE E, su modelo en CyanogenMod se llama *surnia* . Para realizar esta tarea, se ejecuta el comando *breakfast* seguido del *target* a compilar. Este comando descarga ficheros específicos para este dispositivo así como los ficheros fuentes del kernel:

```
$ breakfast surnia
```

Una vez han sido descargados los ficheros específicos para el dispositivo, puede empezar la compilación del código fuente. Antes de iniciarla, es posible activar *ccaché* para hacer que las siguientes compilaciones sean más rápidas [43]. Esto se consigue añadiendo la siguiente línea en el fichero `/.bashrc`:

```
$ export USE_CCACHE=1
```

Luego hay que indicar la cantidad de memoria que estará dedicada a la caché. Esto se realiza ejecutando el siguiente comando (por ejemplo, para 50 GB):

```
$ prebuilts/misc/linux-x86/ccache/ccache -M 50G
```

El último paso consiste en la compilación. Primero se ejecuta el comando *croot* el cual equivale a un *cd* hacia la raíz del código fuente. Después, se llama al compilador con el comando *brunch* seguido del target que se ha especificado anteriormente con el comando *breakfast* :

```
$ croot
$ brunch surnia
```

Este último comando se encarga de compilar todos los módulos. Para el código escrito en el lenguaje C o C++ se utiliza una *CrossToolchain* la cual no solo se encarga de compilar, linkar el código con las librerías y generar los objetos, sino que además produce código para otra plataforma. En este caso para ARM al tratarse de Android. Para la compilación del kernel se requiere de un programa autónomo y no utiliza ninguna librería externa. Los otros módulos, como el framework de Android, servicios de sistema y las aplicaciones se compilan a través del compilador de Java. Finalmente, todas las aplicaciones y ficheros fuente son empaquetados, se crean las imágenes del sistema de ficheros las cuales serán luego instaladas en el dispositivo final. Toda esta información se puede consultar en la wiki de *Embedded Linux* la cuál tiene como espónsor a la *Linux Foundation* [44].

El comando *brunch* además también aprovecha el paralelismo gracias a que al llamar internamente al comando *make* le pasa como parámetro el número de threads a ejecutar en paralelo, cada uno ejecutado en un núcleo distinto [45]. En el caso del ordenador del autor son 8 núcleos. En la Figura 7.10 se observa en la primera línea cómo brunch ejecuta internamente el comando make con el atributo *-j8* para usar todos los núcleos y así aprovechar el paralelismo durante la compilación.

En cuanto al tiempo de compilación, depende principalmente del ordenador desde el cual se realice y de si se trata de la primera compilación o no. En el caso del autor de este proyecto que usa un portátil HP Notebook Envy 15-AK110NS i7-6700HQ, la primera compilación tardó 2 horas 35 minutos en completarse. Destacar que gran parte de este tiempo los 8 núcleos del ordenador estaban procesando a niveles de más del 60 %, y en determinados momentos estaban prácticamente los 8 al 100 % como se observa en la Figura 7.11. La memoria RAM también se ve afectada aumentando en más de 3 GB adicionales su ocupación durante el proceso de compilación <sup>1</sup>. Por lo tanto, queda evidenciada la gran

---

<sup>1</sup>Se ha calculado restando la memoria RAM ocupada durante la compilación menos la memoria RAM ocupada tras la finalización.

```

Command
make -C /home/riordila/android/system -j8 bacon
/usr/bin/X -core :0 -seat seat0 -auth /var/run/lightdm/
/opt/google/chrome/chrome
htop
/opt/google/chrome/chrome --type=renderer --enable-feat
/usr/bin/mongod --config /etc/mongod.conf
/opt/google/chrome/chrome
gala
gnome-screenshot --area
/opt/google/chrome/chrome --type=renderer --enable-feat

```

Figura 7.10: Comando de compilación en make con 8 núcleos

carga de trabajo que supone la compilación cruzada de x86 a ARM. Para las siguientes compilaciones, el tiempo se reduce considerablemente, ya que al hacer uso de la ccaché, solo compila el código nuevo el qual ha sido descargado previamente con el comando:

```
$ repo sync
```

```

+  x  htop
1  [|||||||||||||||||||||||||||||||||||||96.2%] 5  [|||||||||||||||||||||||||||||||||||||98.1%]
2  [|||||||||||||||||||||||||||||||||||||98.7%] 6  [|||||||||||||||||||||||||||||||||||||97.4%]
3  [|||||||||||||||||||||||||||||||||||||96.8%] 7  [|||||||||||||||||||||||||||||||||||||96.2%]
4  [|||||||||||||||||||||||||||||||||||||97.5%] 8  [|||||||||||||||||||||||||||||||||||||98.7%]
Mem[|||||||||||||||||||||||||||||||||4425/7822MB] Tasks: 141, 408 thr; 9 running
Swp[|||||||||||||||||||||||||||||||||0/8030MB] Load average: 10.51 4.56 2.75
Uptime: 00:30:33

PID USER      PRI  NI  VIRT   RES   SHR  S  CPU% MEM%  TIME+  Command
26492 rioardila 21    1 3677M 192M 16980 S 371. 2.5 0:23.35 javac -J-Xmx1024M -source 1.7 -target 1.7 -Xmaxerrs 999
26564 rioardila 21    1 3677M 150M 17112 S 350. 1.9 0:13.90 javac -J-Xmx1024M -source 1.7 -target 1.7 -Xmaxerrs 999
26508 rioardila 21    1 3677M 192M 16980 R 91.7 2.5 0:04.95 javac -J-Xmx1024M -source 1.7 -target 1.7 -Xmaxerrs 999
26507 rioardila 21    1 3677M 192M 16980 R 87.3 2.5 0:04.87 javac -J-Xmx1024M -source 1.7 -target 1.7 -Xmaxerrs 999
26605 rioardila 21    1 3677M 150M 17112 R 79.6 1.9 0:02.81 javac -J-Xmx1024M -source 1.7 -target 1.7 -Xmaxerrs 999
26493 rioardila 21    1 3677M 192M 16980 R 79.0 2.5 0:04.44 javac -J-Xmx1024M -source 1.7 -target 1.7 -Xmaxerrs 999
26603 rioardila 21    1 3677M 150M 17112 R 75.8 1.9 0:02.78 javac -J-Xmx1024M -source 1.7 -target 1.7 -Xmaxerrs 999
26509 rioardila 21    1 3677M 192M 16980 R 75.2 2.5 0:04.54 javac -J-Xmx1024M -source 1.7 -target 1.7 -Xmaxerrs 999
26606 rioardila 21    1 3677M 150M 17112 R 74.5 1.9 0:03.01 javac -J-Xmx1024M -source 1.7 -target 1.7 -Xmaxerrs 999
26566 rioardila 21    1 3677M 150M 17112 R 65.6 1.9 0:02.62 javac -J-Xmx1024M -source 1.7 -target 1.7 -Xmaxerrs 999
26608 rioardila 21    1 3677M 150M 17112 S 39.5 1.9 0:01.69 javac -J-Xmx1024M -source 1.7 -target 1.7 -Xmaxerrs 999
26510 rioardila 21    1 3677M 192M 16980 S 19.7 2.5 0:02.14 javac -J-Xmx1024M -source 1.7 -target 1.7 -Xmaxerrs 999
26324 rioardila 21    1 3942M 122M 17028 S 8.9 1.6 0:07.50 javadoc -encoding UTF-8 @/home/riordila/android/system
24979 rioardila 21    1 117M 101M 16832 D 8.3 1.3 0:03.39 /home/riordila/android/system/out/host/linux-x86/bin/a
20152 rioardila 21    1 7311M 1500M 19680 S 7.6 19.2 1:46.96 java -Dfile.encoding=UTF-8 -Xms2560m -XX:+TieredCompila
1568 root      20    0 627M 92008 72360 S 7.6 1.1 0:59.46 /usr/bin/X -core :0 -seat seat0 -auth /var/run/lightdm/
20181 rioardila 21    1 7311M 1500M 19680 S 5.1 19.2 0:18.16 java -Dfile.encoding=UTF-8 -Xms2560m -XX:+TieredCompila
26501 rioardila 21    1 3677M 192M 16980 S 4.5 2.5 0:00.28 javac -J-Xmx1024M -source 1.7 -target 1.7 -Xmaxerrs 999
26356 rioardila 21    1 3942M 122M 17028 S 3.8 1.6 0:00.81 javadoc -encoding UTF-8 @/home/riordila/android/system
12464 rioardila 21    1 610M 583M 2156 D 3.8 7.5 0:36.36 make -C /home/riordila/android/system -j8 bacon

```

Figura 7.11: Salida del comando *htop* durante la compilación de Android.

Una vez finalizada la compilación, se pueden obtener los ficheros generados accediendo al directorio `out/target/product/<device-name>/`. Aquí se ubican las imágenes de boot, recovery, ramdisk, el kernel, el sistema de ficheros, así como el .zip que se importa en el modo recovery para instalar la nueva ROM en el dispositivo. Si se desea realizar un borrado completo de los ficheros generados, basta con eliminar el contenido del directorio `out`. En

la Figura 7.12 puede observarse el contenido de este directorio después de haber realizado tres compilaciones en fechas distintas.

```

rioardila@marenosturm2:~/android/system/out/target/product/surnia$ ls
android-info.txt          cm_surnia-ota-d46c020717.zip  ramdisk.img
boot.img                  data                          ramdisk-recovery.cpio
cache                     dt.img                        ramdisk-recovery.img
cache.img                 fake_packages                 recovery
clean_steps.mk           gen                            recovery.id
cm-13.0-20160928-UNOFFICIAL-surnia.zip  install                       recovery.img
cm-13.0-20160928-UNOFFICIAL-surnia.zip.md5sum  installed-files.txt          root
cm-13.0-20161002-UNOFFICIAL-surnia.zip  kernel                         symbols
cm-13.0-20161002-UNOFFICIAL-surnia.zip.md5sum  obj                            system
cm-13.0-20161104-UNOFFICIAL-surnia.zip  ota_override_device          system.img
cm-13.0-20161104-UNOFFICIAL-surnia.zip.md5sum  ota_script_path              userdata.img
cm_surnia-ota-2e2a883995.zip             ota_temp                      previous_build_config.mk
cm_surnia-ota-3fbb1d8f22.zip

```

Figura 7.12: Contenido del directorio out/target/product/surnia.

### 7.5.3. Ejecución

Una vez se han generado los ficheros de salida tras el proceso de compilación, ya es posible de instalar la nueva ROM en el dispositivo. Para ello, se implementan dos maneras distintas de ejecutarlo: en el dispositivo físico y en los dos emuladores:

#### En dispositivo físico

Para ejecutar la nuevo ROM en el dispositivo físico, primero hay que copiar la ROM en la memoria interna o en la tarjeta SD. A continuación, se debe acceder al modo *Recovery* desde el terminal móvil. En el caso del Motorola LTE E si se pulsan todas las teclas al mismo tiempo (cuando el dispositivo se encuentra apagado), se accede a este modo. La otra manera consiste en hacerlo vía ADB. Para ello, conectándolo por cable USB al ordenador y con el terminal encendido, se debe escribir el comando:

```
$ adb reboot recovery
```

Después, una vez en el menú principal del modo recovery, se debe hacer un borrado completo de la memoria (*Factory reset*). Para ello se accede a *Wipey* pulsa en *Swipe to Factory Reset* como se observa en la Figura 7.13. Después, ya es posible de instalar la nuevo ROM pulsando en *Install* desde el menú principal. En el listado que aparece, se selecciona el paquete que contiene la ROM y pulsa en *Swipe to Confirm Flash*.

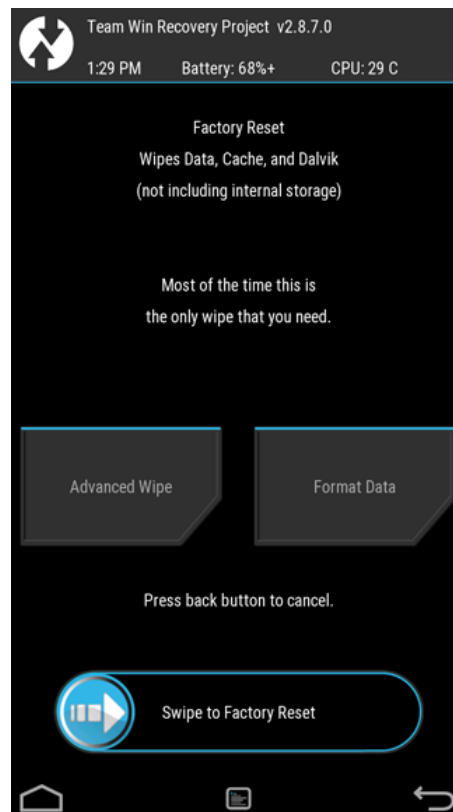


Figura 7.13: Pantalla de Wipe con el programa de recovery TWRP.

### En emulador

En este caso hay que tener en cuenta que el emulador como tal no deja de ser un dispositivo más y, por tanto, el sistema operativo que se le instale debe estar preparado para funcionar en ese dispositivo. En el caso del emulador de Android, debe usarse la imagen genérica del proyecto AOSP o una ROM preparada para este dispositivo. CyanogenMod también dispone de versiones para ser usadas en dispositivos de Google (emulador incluido), por ejemplo para el *Nexus 9 (LTE)*[46].

Para ejecutar el emulador de Android con una ROM distinta a la que viene por defecto, se puede llamar vía comando pasándole por parámetro la ruta de la nueva imagen de sistema así como la imagen de *ramdisk*:

```
$ emulator -avd MyPhone -system out/target/product/surnia/system.img  
-ramdisk out/target/product/surnia/ramdisk.img &
```

## 7.6. Mejoras implementadas

En esta sección se explican las distintas mejoras realizadas sobre el laboratorio virtual, ya sea mediante la inclusión de herramientas o la implementación de scripts.

### 7.6.1. Cross-compiling

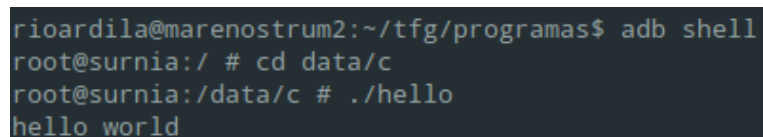
Una de las pruebas realizadas consiste en probar de ejecutar un programa escrito en lenguaje C en el dispositivo Android. Hay que, por tanto, compilar el código de tal manera que sea ejecutado en una arquitectura distinta a la del ordenador. Como el dispositivo físico usa arquitectura ARM, se ha probado a compilar el programa para esta arquitectura y comprobar su ejecución posterior en el dispositivo. Por defecto, Ubuntu no dispone de herramientas de compilación cruzada en C. Las herramientas de compilación y librerías necesarias se obtienen con los siguientes comandos:

```
$ sudo apt-get install libc6-armel-cross libc6-dev-armel-cross
$ sudo apt-get install binutils-arm-linux-gnueabi
$ sudo apt-get install libncurses5-dev
$ sudo apt-get install gcc-arm-linux-gnueabi
$ sudo apt-get install g++-arm-linux-gnueabi
```

Una vez instaladas las dependencias, es posible compilarlo a ARM con el siguiente comando:

```
$ arm-linux-gnueabi-gcc programm.c -o programm
```

A continuación debe copiarse al dispositivo el ejecutable generado. Para ello se ha usado la herramienta *adb push*, la cual permite mover archivos del ordenador al terminal Android. Una vez copiado, es posible abrir una línea de comandos con el comando *adb shell*. A partir de ahí, se accede al directorio donde se almacenó el ejecutable y ya es posible ejecutarlo como se observa en la Figura 7.14.



```
rioardila@marenostrom2:~/tfg/programas$ adb shell
root@surnia:/ # cd data/c
root@surnia:/data/c # ./hello
hello world
```

Figura 7.14: Ejecución de HelloWorld compilado para ARM en dispositivo físico mediante *adb shell*.

Esta herramienta ha sido incluida en el laboratorio virtual.



### 7.6.2. Script de automatización

Se ha implementado un script capaz de automatizar una serie de tareas repetitivas. En este caso, el script una vez ejecutado es capaz de realizar la compilación cruzada del programa en C para arquitectura ARM, copiar el ejecutable generado en el dispositivo a través de *adb push* (tanto físico como virtual), y ejecutarlo vía *adb shell*.

```
#!/bin/bash

#check number of parameters
if [ $# -ne 2 ]; then
    echo $0: usage: compile-push-arm.sh input output
    exit 1
fi

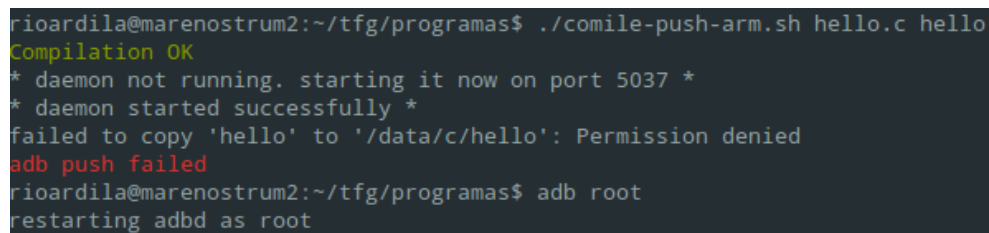
#compile programm
arm-linux-gnueabi-gcc -static "$1" -o "$2"
if [ $? -ne 0 ]; then
    echo "$(tput setaf 1)Compilation failed"
    tput sgr0
    exit 1
else
    echo "$(tput setaf 2)Compilation OK"
    tput sgr0
fi

#adb push to device /data/c (creates c folder if it doesn't exist)
adb shell mkdir -p /data/c
adb push "$2" /data/c
if [ $? -ne 0 ]; then
    echo "$(tput setaf 1)adb push failed"
    tput sgr0
    exit 1
else
    echo "$(tput setaf 2)adb push OK"
    tput sgr0
fi

#run programm from adb shell
echo Program output:
```

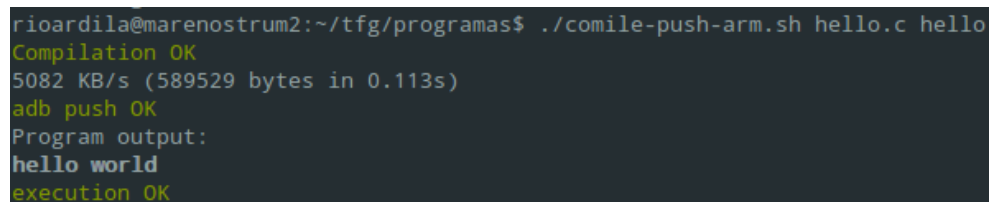
```
tput bold
adb shell /data/c/"$2"
tput sgr0
if [ $? -ne 0 ]; then
    echo "$(tput setaf 1)execution failed"
    tput sgr0
    exit 1
else
    echo "$(tput setaf 2)execution OK"
    tput sgr0
fi
```

Hay que tener en cuenta que para que el código anterior funcione, hay que ejecutar la herramienta *adb* en modo usuario *root* para que se pueda acceder al directorio *data* y así poder almacenar y ejecutar en él. Sino, el script muestra el mensaje de error de la Figura 7.15. Si ADB se ha iniciado en modo root, el script se ejecutará correctamente como se muestra en la Figura 7.16.



```
rioardila@marenosturum2:~/tfg/programas$ ./comile-push-arm.sh hello.c hello
Compilation OK
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
failed to copy 'hello' to '/data/c/hello': Permission denied
adb push failed
rioardila@marenosturum2:~/tfg/programas$ adb root
restarting adbd as root
```

Figura 7.15: Mensaje de error mostrado al no haber inicializado ADB en modo root previamente.



```
rioardila@marenosturum2:~/tfg/programas$ ./comile-push-arm.sh hello.c hello
Compilation OK
5082 KB/s (589529 bytes in 0.113s)
adb push OK
Program output:
hello world
execution OK
```

Figura 7.16: Ejecución correcta del script tras inicialización de ADB en modo root.

Una mejora del código anterior sería la introducción de la instrucción *adb root* en el script para que lo iniciara antes de llamar a la instrucción *adb push* .

Otro posible mejora del código anterior sería la de dar la posibilidad al usuario de escoger el tipo de arquitectura a la hora de compilar (ARM o x86). Para ello, habría que modificar la sección de compilación de la siguiente manera:

```
#Choose architecture and compile programm
echo "Choose architecture (1. ARM / 2. x86)."
read -r arc
if [$arc -eq 1]; then
    arm-linux-gnueabi-gcc -static "$1" -o "$2"
else if [$arc -eq 2]; then
    gcc "$1" -o "$2"
else
    echo "$(tput setaf 1)Chosen option is not correct"
    tput sgr0
    exit 1
fi

#Check if compilation succeeded
if [ $? -ne 0 ]; then
    echo "$(tput setaf 1)Compilation failed"
    tput sgr0
    exit 1
else
    echo "$(tput setaf 2)Compilation OK"
    tput sgr0
fi
```

Ambas mejoras han sido incluidas en el código final del script y añadidas al laboratorio virtual.

## 7.7. Creación del laboratorio virtual

En esta sección se proponen distintas maneras de juntar todas las herramientas de tal manera que el laboratorio virtual quede unificado y sea relativamente sencillo tener el entorno de trabajo preparado para empezar a testear con el código fuente de Android.

### 7.7.1. Sugerencia 1: Imagen de Ubuntu

Se propone crear una imagen clonada del sistema operativo Ubuntu <sup>2</sup> con todas las herramientas ya instaladas y configuradas. Además, el código fuente de Android está ya descargado y se le ha realizado una primera compilación. Para realizar la clonación, se propone usar el software *Clonezilla Live* [47]. Este software se encarga de clonar el sistema operativo de manera que se obtiene como resultado una copia exacta del estado actual del mismo. En este caso, este software resulta de gran utilidad ya que permite realizar un despliegue rápido y sin necesidad de tener que instalar algún software ya que todo está incluido en la imagen.

Para realizar la copia del sistema operativo, es necesario instalar el Clonezilla Live en un soporte físico (ej: USB, CD-ROM) para que sea iniciado al arrancar el ordenador como si se tratara de un sistema operativo. Una vez en el menú principal, es posible realizar una clonación de todo el contenido del disco duro o bien de una partición en concreto. Para este proyecto, se utiliza la opción *savedisk* ya que es la más sencilla de implementar, aunque también sería posible hacer la copia de particiones. Después de realizar la clonación, es posible hacer la restauración en un disco duro con la opción *restoredisk*. En la Figura 7.17 se puede observar una captura del menú principal y en la 7.18 el progreso de clonación para un dispositivo de 5GB.

```
Clonezilla: Select mode
*Clonezilla is free (GPL) software, and comes with ABSOLUTE NO WARRANTY*
This software will overwrite the data on your hard drive when restoring! It is recommended to
backup important files before restoring!***
///Hint! From now on, if multiple choices are available, you have to press space key to mark
your selection. An asterisk (*) will be shown when the selection is done///
Select mode:

savedisk      Save_local_disk_as_an_image
saveparts    Save_local_partitions_as_an_image
*restoredisk  Restore_an_image_to_local_disk
restoreparts Restore_an_image_to_local_partitions
recovery-iso-zip Create_recovery_Clonezilla_live
exit         Exit. Enter command line prompt

<Ok>                <Cancel>
```

Figura 7.17: Menú principal del software Clonezilla Live

El tiempo de clonación es relativamente corto, siempre dependiendo de la velocidad de escritura/lectura del disco duro y del soporte físico (USB o CD-ROM). Se han realizado pruebas en el ordenador del autor y se ha obtenido que la velocidad media es de 16 GB/-min. Entonces, para un disco duro de 500 GB el tiempo total de creación de la imagen sería de 31 minutos aproximadamente. El proceso de restauración tarda prácticamente el

<sup>2</sup>Ubuntu porque es el sistema operativo empleado en el laboratorio virtual.



### 7.7.4. Evaluación

Cada una de las tres sugerencias tiene sus ventajas e inconvenientes. A continuación se analizan para cada una:

#### – Imagen de Ubuntu

- **Ventajas:**

- Rápida y fácil puesta en marcha: solo es necesario usar el programa Clonezilla para restaurar la imagen en el disco duro y a partir de ahí ya se puede empezar a usar.
- Es posible de almacenar la imagen en un disco en red como *sshfs* [48] o *samba* [49]: esto hace que la imagen quede centralizada en un servidor.

- **Inconvenientes:**

- Al cabo de un tiempo el software estará desactualizado y sería necesario realizar una nueva imagen cada cierto tiempo.
- Los alumnos no aprenderían el proceso de instalación y configuración del laboratorio virtual.
- El dispositivo físico final debe decidirse antes de realizar la imagen, ya que el código fuente difiere de un terminal a otro.

#### – Guía de instalación

- **Ventajas:**

- Los alumnos aprenderían paso por paso cómo implementar el laboratorio virtual.
- Al no tener el código fuente de Android precompilado, es posible de escoger el terminal móvil después de la implementación del laboratorio.

- **Inconvenientes:**

- Requiere más tiempo de preparación al tener que ir paso a paso instalado y configurando cada software.
- La descarga y primera compilación del código fuente de Android es de 4-6 horas aproximadamente.

#### – Guía de instalación con código fuente

- **Ventajas:**

- Igual que la segunda sugerencia pero esta vez el tiempo de preparación se reduce al ya tener el código fuente descargado y precompilado.

- **Inconvenientes:** Debe decidirse previamente qué terminal móvil se usará, ya que el código fuente está precompilado para ese dispositivo.

Evaluando los pros y los contras de las tres sugerencias, parece que la opción más viable es la última: realizar una guía de instalación con el código fuente ya descargado y precompilado. De esta manera, los alumnos serán capaces de preparar el entorno de trabajo por ellos mismos y el tiempo se reducirá al no tener que descargar ni realizar una primera compilación del código fuente.

# Capítulo 8

## Prácticas de laboratorio

### 8.1. Consideraciones generales

En esta segunda parte del proyecto se proponen e implementan una serie de prácticas de laboratorio con el propósito de que los estudiantes sean capaces de realizar modificaciones en el código fuente del kernel de Android.

Las prácticas que se sugieren son:

1. **Instalación del entorno virtual:** Mediante una guía se indican los pasos necesarios para realizar desde cero la instalación del laboratorio virtual con todas las herramientas necesarias para su correcto funcionamiento.
2. **Governor de Android:** Con esta práctica se pretende usar un perfil de CPU diferente al que viene por defecto. Además, se pretenden sacar estadísticas de rendimiento.
3. **Añadir app de sistema:** Consiste en la inclusión de una aplicación en Android y en lenguaje C en el propio código de sistema de manera que al realizar un reinicio de fábrica ésta siga estando disponible.
4. **Crear módulo de kernel:** Al igual que en una distribución Linux de escritorio, se pretende demostrar que en Android también es posible crear y cargar un módulo en el kernel.
5. **Sistema de ficheros F2FS:** Se trata de un sistema de ficheros optimizado para ser usado en memorias tipo flash. Se pretende formatear la partición de datos usando este sistema de ficheros y realizar estadísticas de rendimiento respecto al formato ext4.



## 8.2. Práctica 1: Instalación del entorno virtual

Esta práctica se basa en la realización de la tercera sugerencia de implementación del laboratorio virtual, tal y como se describe en el apartado 7.7.3. La idea principal es la de seguir un guión con la descripción de todas las herramientas necesarias para la instalación completa del laboratorio virtual. Por otra parte, como se sugiere, el código fuente ya ha sido previamente descargado y precompilado para ahorrar tiempo al estudiante. Una vez todas las herramientas han sido instaladas, ya es posible de realizar una recompilación y comprobar su correcto funcionamiento tanto en el dispositivo físico como en el emulador.

## 8.3. Práctica 2: Governor de Android

En esta práctica se propone añadir un *governor* personalizado al kernel de Android. Un governor en Android es un perfil de comportamiento de la CPU, es decir, indica a la CPU cómo debe actuar ante determinadas situaciones. Los governor por defecto que vienen incluidos en cualquier sistema Android son: *bajo demanda*, *ahorro de energía*, *rendimiento*, *conservador* y *espacio de usuario*. Todos estos perfiles governor son muy básicos y no aportan funcionalidades avanzadas que podrían ser aprovechadas por la CPU. Cada uno de estos perfiles implica un comportamiento distinto de la CPU:

- **Bajo demanda:** Es el perfil por defecto de todo dispositivo Android. Lo que hace es aumentar la frecuencia de la CPU cuando se inicia la aplicación y una vez ha sido cargada la disminuye. El problema es que la aplicación ralentiza una vez ha sido cargada.
- **Ahorro de energía:** La CPU siempre trabaja a la mínima frecuencia a costa de obtener un bajo rendimiento.
- **Rendimiento:** Al contrario que el perfil de ahorro de energía, en éste la CPU trabaja a su máxima frecuencia a costa de un consumo elevado de la batería.
- **Conservador:** Está a un nivel intermedio entre ahorro de energía y rendimiento. Aporta un rendimiento aceptable sin un gran consumo de la batería.
- **Espacio de usuario:** Es un governor que se configura manualmente. Por defecto está configurado con el perfil de ahorro de energía. Este governor rara vez es usado.

Para esta práctica se propone, por tanto, añadir nuestro propio perfil governor con el objetivo de mejorar los aspectos de *performance* y consumo de batería. El governor consiste en un fichero escrito en C el cual se encuentra en

```
kernel_source/drivers/cpufreq/
```

Este directorio contiene todos los perfiles governor que existen en el kernel. En la Figura 8.1 se listan los ficheros governor disponibles para la imagen Surnia de Android.

```
rioardila@marenostrium2:~/android/system/kernel/motorola/msm8916/drivers/cpufreq$ ls | grep cpufreq_
cpufreq_conservative.c
cpufreq_governor.c
cpufreq_governor.h
cpufreq_interactive.c
cpufreq_ondemand.c
cpufreq_performance.c
cpufreq_persistent_stats.c
cpufreq_powersave.c
cpufreq_stats.c
cpufreq_userspace.c
ppc_cbe_cpufreq_pervasive.c
ppc_cbe_cpufreq_pmi.c
```

Figura 8.1: Directorio con todos los ficheros governor para Surnia.

A continuación se propone añadir un governor distinto a los que aparecen listados. En concreto, una llamado *smartassv2*. Este governor intenta usar siempre la frecuencia ideal y subir de forma bastante agresiva hasta esa frecuencia, para después bajar más suavemente. Usa diferentes frecuencias ideales para perfiles de pantalla apagada/encendida, llamados *awake\_ideal\_freq* y *sleep\_ideal\_freq*. Este governor baja de frecuencia de CPU muy rápidamente (para alcanzar cuanto antes la *sleep\_ideal\_freq*) mientras la pantalla está apagada, y sube de frecuencia de la CPU rápidamente hasta la *awake\_ideal\_freq* cuando la pantalla se enciende. Por tanto, el objetivo de este governor es llegar a un equilibrio entre rendimiento y batería.

Primero de todo hay que añadir el fichero escrito en C en el directorio anterior. A continuación hay que editar el fichero *Kconfig* que se ubica en el mismo directorio. Este fichero contiene la configuración del menú de configuración del kernel. Añadiendo las siguientes líneas se le da la posibilidad al usuario de escoger el nuevo governor para compilarlo junto al kernel:

```
config CPU_FREQ_GOV_SMARTASS2
    tristate "'smartassV2' cpufreq governor"
    depends on CPU_FREQ
    help
    'smartassV2' - a "smart" optimized governor!
```

Además, hay que dar la posibilidad de que ese governor sea escogido como governor por defecto. Esto se consigue añadiendo el siguiente código en el mismo fichero *Kconfig*:

```
config CPU_FREQ_DEFAULT_GOV_SMARTASS2
bool "smartass2"
select CPU_FREQ_GOV_SMARTASS2
help
Use the CPUFreq governor 'smartassV2' as default.
```

A continuación hay que editar el fichero Makefile ubicado en el mismo directorio, indicándole que debe compilar el nuevo governor:

```
obj-$(CONFIG_CPU_FREQ_GOV_SMARTASS2) += cpufreq_smartass2.o
```

Después hay que editar el fichero *cpufreq.h* ubicado en

```
kernel_source/includes/linux
```

Y añadir el nuevo governor a la librería de frecuencias de la CPU:

```
#elif defined(CONFIG_CPU_FREQ_DEFAULT_GOV_SMARTASS2)
extern struct cpufreq_governor cpufreq_gov_smartass2;
#define CPUFREQ_DEFAULT_GOVERNOR (&cpufreq_gov_smartass2)
```

Una vez se tiene el kernel configurado para usar el nuevo governor por defecto, ya es posible de recompilarlo e instalarlo en el dispositivo Android.

## 8.4. Práctica 3: Añadir app de sistema

Para esta práctica se propone crear y añadir nuestra propia aplicación para que forme parte de las aplicaciones de sistema. Esto implica que aunque se realice una restauración de fábrica la aplicación seguirá estando disponible en la imagen de Android, ya que formará parte del espacio de sistema y no de usuario. Este proceso solo es posible de realizar si el dispositivo Android ha sido rooteado previamente ya que la aplicación quedará instalada dentro de la partición *system* la cual por motivos de seguridad no dispone de permisos de escritura para el usuario estándar de Android.

Se añadirán dos aplicaciones distintas. Una de ellas será una aplicación en Android y la otra será un programa escrito en C. Para este segundo programa, como se verá más adelante, hay que indicarle al compilador que la arquitectura final es distinta a la del propio

ordenador (se usará una herramienta de cross-compiling para tal propósito, la cual ya ha sido incluida a propósito en el laboratorio virtual).

Por otro lado, ambas aplicaciones serán instaladas y ejecutadas tanto en el dispositivo físico como en los emuladores del laboratorio virtual.

### 8.4.1. Aplicación Android

Para la aplicación en Android es posible usar cualquiera que nos interese tener por defecto en la imagen. Se ha pensado que podría ser útil tener una instalada una aplicación para controlar el dispositivo en remoto de manera que si es olvidado en cualquier sitio o ha sido robado, aunque se le realice un borrado completo de fábrica, la aplicación seguirá estando disponible y se dispondrá de acceso para que el dispositivo sea controlado remotamente.

La aplicación escogida se llama *Control Ur Phone Remotely LITE\_v1.1\_apkpure.com.apk*, aunque es posible usar cualquier otra parecida. Entre algunas de las características de las que disponen estas aplicaciones, cabe destacar la posibilidad de ubicar geográficamente al dispositivo, capturar imágenes, etc.

Para realizar la instalación de la aplicación en modo sistema deben seguirse estos pasos:

1. Descargarse el fichero APK de la aplicación en el ordenador. Cualquier fichero APK capaz de instalarse de la manera habitual (en modo usuario) puede ser usado para ser instalado en modo sistema.
2. Conectar el dispositivo Android al ordenador con el cable USB. Ejecutar el daemon de ASDB en modo root y comprobar que el dispositivo está disponible ejecutando el comando de adb:

```
$ adb root
$ adb devices
```

```
rioardila@marenostrom2:~/android/system$ adb root
restarting adbd as root
rioardila@marenostrom2:~/android/system$ adb devices
List of devices attached
TA36405JC0      device
```

Figura 8.2: Ejecución exitosa de *adb root* y *adb devices*.

3. Una vez el dispositivo está disponible, hay que proceder a copiar el fichero APK en el directorio de sistema, asignarle los permisos adecuados y reiniciar el dispositivo. Suponiendo que el APK se llama `file.apk`, los comandos a ejecutar son:

```
$ adb remount
$ adb push file.apk /system/app/
$ adb shell chmod 644 /system/app/file.apk
$ adb reboot
```

4. Una vez se haya reiniciado el terminal, se puede observar como la aplicación aparece ya instalada (Figura 8.3).

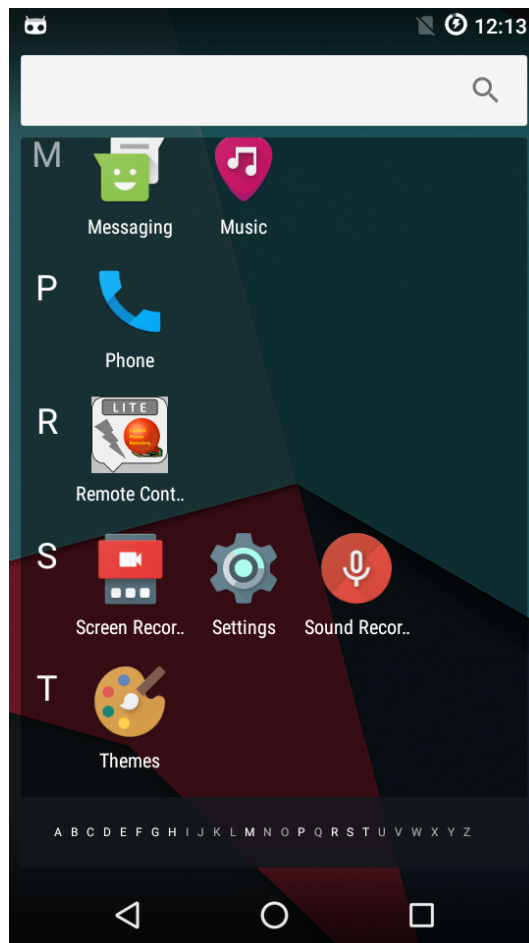


Figura 8.3: Captura de pantalla donde se observa la aplicación *Remote Control* instalada.

Si se realiza una restauración de fábrica o un *wipe* completo desde el menú recovery, es decir, un borrado de la partición de usuario, la aplicación seguirá estando disponible ya

que está ubicada dentro de la partición de sistema.

No obstante, el proceso de instalación anterior solo sirve para un dispositivo el cual ya ha sido flasheado previamente, es decir, se le ha instalado una ROM como el caso de CyanogenMod. Es interesante también ser capaz de añadir la aplicación como parte del código fuente de Android para, por ejemplo, que sea distribuida como aplicación de sistema de una ROM. Para ello, basándose en una aplicación que no ha sido compilada previamente, deberían realizarse los siguientes pasos:

1. Crear o obtener el código fuente de una aplicación para Android. El proceso de compilación final de esta aplicación será distinto al usualmente realizado en *Android Studio* o *Eclipse*. Por eso, una vez ha sido creado el código fuente, es necesario realizar una limpieza del proyecto de manera que solo queden los ficheros fuentes y no ficheros generados debido a alguna compilación del proyecto.
2. Una vez se tiene el código fuente del proyecto, debe copiarse el proyecto dentro del directorio *packages/apps*. En este directorio se ubican todas las aplicaciones que serán instaladas con la imagen. En la Figura 8.4 se observa el contenido de este directorio para el código fuente de CyanogenMod 13.0.

```

rioardila@marenostrom2:~/android/system/packages/apps$ ls
AudioFX          CMFileManager   FMRadio         PackageInstaller Tag
BasicSmsReceiver CMUpdater       Gallery2        PhoneCommon      Terminal
Bluetooth        CMWallpapers    Gello           Profiles          ThemeChooser
BluetoothExt     Contacts        HTMLViewer      Provision         Trebuchet
Browser          ContactsCommon InCallUI        Screenshot       TvSettings
Calendar         DeskClock       KeyChain        Settings          UnifiedEmail
Camera2          Dialer          LockClock       SetupWizard      WallpaperPicker
CarrierConfig    Eleven          ManagedProvisioning SmartCardService
CellBroadcastReceiver Email            Messaging       Snap
CertInstaller    ExactCalculator Nfc             SoundRecorder
CMBugReport      Exchange        OneTimeInitializer Stk

```

Figura 8.4: Contenido del directorio *packages/apps* que forma parte del código fuente de CyanogenMod 13.0.

3. A continuación, hay que crear el fichero *makefile* necesario para indicar al compilador cómo realizar la compilación de la aplicación. Para ello, dentro del directorio de la aplicación, debe crearse el fichero *Android.mk* con el siguiente contenido (suponiendo que la aplicación se llama *RemoteDemo*):

```

LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)

#Build all java files in the java subdirectory
LOCAL_SRC_FILES := $(call all-subdir-java-files)

```

```
#Name of the APK to build
LOCAL_PACKAGE_NAME := RemoteDemo
```

```
#Tell it to build an APK
include $(BUILD_PACKAGE)
```

4. Después, debe añadirse la aplicación al listado de *PRODUCT\_PACKAGES* . Para ello, debe editarse el fichero *core.mk* ubicado en el directorio:

```
/build/target/product/
```

Y añadir el nombre de la aplicación como una entrada más dentro de la variable *PRODUCT\_PACKAGES* .

5. Tras esto, ya es posible de realizar la compilación completa del código fuente con el comando *brunch surniay* una vez finalizada se habrá generado un nuevo fichero *RemoteDemo.apk* dentro del directorio de salida:

```
out/target/product/surnia/system/app/
```

Una vez la nueva aplicación ha sido incluida como parte de la nueva imagen de Android, cuando se instale esa imagen en cualquier dispositivo la aplicación añadida será instalada junto a las demás aplicaciones de sistema.

A continuación se muestra cómo testearlo en el AVD de Android. Para ello, debe ejecutarse el emulador llamando a la nueva imagen de sistema que se ha creado (*MyPhone* hace referencia al nombre que se le ha dado al emulador):

```
$ emulator -avd MyPhone -system out/target/product/surnia/system.img
-ramdisk out/target/product/surnia/ramdisk.img &
```

Una vez iniciado el emulador, se observa que la aplicación aparece en el menú de aplicaciones.

### 8.4.2. Programa en C

También es posible de almacenar un programa en C dentro de la partición de sistema para que, al igual que en el caso de la aplicación en Android, éste no pueda ser borrado si se realiza un borrado completo de la memoria de usuario. Hay que tener en cuenta

que el programa que se cree no aparecerá listado como una aplicación más, sino que solo se podrá ejecutar vía la línea de comandos (*adb shell*) o si es llamado por otra aplicación.

Los pasos a seguir para instalar el programa en C dentro de la partición de sistema son:

1. Compilar el fichero en C usando para ello un herramienta de cross-compiling como la que se incluye en el laboratorio virtual. Esto hará que el programa generado sea capaz de funcionar en otra arquitectura distinta. Para ello, imaginando que el fichero se llama *programm.c*, se debe ejecutar:

```
$ arm-linux-gnueabi-gcc programm.c -o programm
```

2. Una vez se ha generado el programa para arquitectura ARM, ya se puede copiar al dispositivo en el directorio */system/bin*:

```
$ adb remount
$ adb push programm /system/bin/
$ adb shell chmod 644 /system/bin/programm
$ adb reboot
```

3. Una vez copiado y reiniciado el dispositivo, si se le realiza un borrado de la memoria de usuario, se observará que el programa sigue estando disponible y es capaz de ser ejecutado.

En el caso que se quisiera incluir el programa como parte de la imagen de Android, el procedimiento es más sencillo que en el caso de una aplicación en Android. Simplemente debe copiarse el programa ya compilado en el directorio de salida del código fuente:

```
$ cp programm out/target/product/surnia/system/bin/
```

Una vez ha sido copiado, ya es posible de realizar la compilación completa del código fuente de Android y de esta manera el programa será incluido dentro de la imagen *system.img*. Para testearlo en el emulador AVD, igual que anteriormente, se debe ejecutar el comando:

```
$ emulator -avd MyPhone -system out/target/product/surnia/system.img
-ramdisk out/target/product/surnia/ramdisk.img &
```



```
rioardila@marenostrom2:~/tfg/programas$ adb push hello /system/bin
3100 KB/s (589529 bytes in 0.185s)
rioardila@marenostrom2:~/tfg/programas$ adb shell /system/bin/hello
hello world
```

Figura 8.5: Copia y ejecución del programa en C en el directorio `/system/bin` de Android.

## 8.5. Práctica 4: Crear módulo de kernel

Para esta práctica se propone crear un nuevo módulo para que sea incluido al kernel de Android, compilarlo junto con todo el código fuente y comprobar su correcto funcionamiento.

El kernel de Android por defecto no tiene activada la opción de cargar módulos. Hay dos maneras de activar esta opción: o bien buscando la opción y habilitándola mediante el asistente *menuconfig* el cual al guardar los cambios generará un fichero de salida llamado *.config*, o bien generar este fichero por defecto y editarlo posteriormente. En el caso de querer usar la segunda alternativa, para generar este fichero para nuestro modelo Surnia, se debe lanzar el comando:

```
$ make ARCH=arm surnia_defconfig
```

Después hay que editar este fichero cambiando la línea

```
# CONFIG_MODULES is not set
```

por

```
CONFIG_MODULES=y
```

Una vez el fichero ha sido editado, ya es posible guardarlo y compilar el código fuente de Android con el nuevo kernel con soporte para la carga de módulos.

A continuación, se ha creado un módulo muy sencillo llamado *my\_module.c* que muestra un mensaje cuando es cargado en el kernel y cuando se descarga. El código fuente es:

```
include"linux/module.h"
include"linux/kernel.h"
//replace the "" with angular brackets
int init_module(void)
{
```

```
    printk(KERN_INFO "Hello world\n");
    return 0;
}

void cleanup_module(void)
{
    printk(KERN_INFO "Hello world\n");
}
```

El contenido del fichero Makefile necesario para compilar el módulo es el siguiente:

```
obj-m += my_module.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Tras esto, ya es posible de compilar el módulo mediante el comando:

```
$ arm-none-linux-gnueabi- ARCH=arm make
```

Se habrá generado un fichero *my\_module.ko*. A continuación, ya es posible de inyectar el nuevo módulo creado en el kernel. Para ello, primero debe ejecutarse el kernel generado anteriormente con la opción de carga de módulos activada en el dispositivo. En el caso del dispositivo físico, simplemente debe flasharse la nueva imagen para que se cargue la nueva ROM". En el caso del emulador Android, se le puede indicar que cargue el nuevo kernel en vez del habitual. Para ello, se ejecutará el AVD de Android con el comando:

```
$ emulator -avd MyPhone -system out/target/product/surnia/system.img
-ramdisk out/target/product/surnia/ramdisk.img
-kernel kernel/motorola/msm8916/arch/arm/boot/zImage
-data out/target/product/surnia/userdata.img
-verbose -show-kernel &
```

Una vez el emulador está corriendo (en el caso de realizarse en el dispositivo físico, el proceso es el mismo), debe copiarse el nuevo módulo con el comando:

```
$ adb push my_module.ko /data/local
```

Se abre una shell con el dispositivo y se ejecutan los comandos:

```
$ adb shell
$ cd data/local
$ insmod my_module.ko
$ rmod my_module
$ dmesg
```

Explicación de cada comando:

- **insmod**: Comando que carga el módulo en el kernel.
- **rmod**: Elimina el módulo del kernel.
- **dmseg**: Lista el búfer de mensajes del kernel. Contiene una gran cantidad de mensajes generados durante el arranque del sistema y otros mensajes de depuración de aplicaciones.

Al ejecutar el último comando *dmseg* es posible observar la salida generada al insertar y quitar el módulo como se ve en la Figura 8.6.

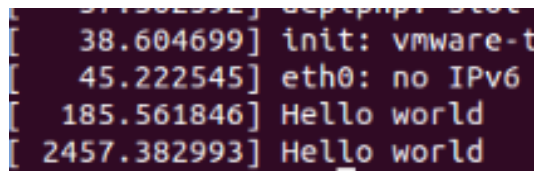
A screenshot of a terminal window showing the output of the 'dmesg' command. The output consists of four lines of kernel messages, each starting with a timestamp in square brackets. The messages are: '[ 38.604699] init: vmware-t', '[ 45.222545] eth0: no IPv6', '[ 185.561846] Hello world', and '[ 2457.382993] Hello world'. The text is white on a dark background.

Figura 8.6: Salida del comando *dmseg* donde se observa el string Hello World.

## 8.6. Práctica 5: Sistema de ficheros F2FS

Como se muestra en el apartado 6.6, el sistema de ficheros predeterminado en Android es el ext4. No obstante, como cualquier distribución basada en Linux, es posible de usar otro sistema de ficheros. Para esta práctica, se propone usar F2FS y realizar pruebas de rendimiento entre ext4 y F2FS para ver si el cambio de sistema de ficheros aporta una mejora de rendimiento al dispositivo o no.

Para proceder a instalar el sistema de ficheros F2FS debemos escoger sobre qué partición realizar la instalación. Una opción sería cambiar el sistema de ficheros de toda la memoria interna pero para en ese caso habría formatearla toda y copiar de nuevo e nuevo las

imágenes con el nuevo sistema de archivos. Otra opción que es tan sólo hacer el cambio sobre la partición */data* y */cache*, de esta manera solo se borran los datos de usuario y no es necesario reinstalar el sistema. Para esta práctica se propone usar el segundo método.

Antes de realizar el cambio de sistema de ficheros EXT4 a F2FS (o a la inversa), se han descargado una serie de benchmarks que serán de gran utilidad para comparar ambos sistemas de ficheros. Los benchmarks que se usan son (todos están disponibles en Google Play):

- **RL Bench:** Está dedicado a realizar pruebas sobre SQLite, realizando una serie de *queries* (consultas) y calculando el tiempo de procesado de todas ellas.
- **CF Bench:** Es un benchmark de CPU y memoria el cual soporta sistemas multi-núcleo.
- **OxBenchmark:** Es un benchmark genérico y muestra los resultados en formato de texto.
- **AnTuTu Benchmark:** Es el más popular por el momento. Cubre el rendimiento de la CPU, RAM, GPU e IO.
- **Quadrant Benchmark:** Benchmark parecido al anterior pero menos popular.

Todas estas herramientas benchmark han sido descargadas e instaladas en el dispositivo físico de Android. A continuación, se ejecutan todos los benchmark para el sistema de ficheros actual para, más adelante, realizar lo mismo con el nuevo sistema ya instalado. Para esta práctica, el dispositivo Android ya dispone del sistema F2FS, por lo que el cambio que se realizará es desde F2FS a EXT4.

Una vez que se han ejecutado y guardado los resultados de todos los benchmark, se puede proceder a cambiar el formato de F2FS a EXT4. Para ello, primero de todo hay que entrar en modo recovery pulsando todos los botones (en el caso del Motorola LTE E) o si está conectado por cable USB mediante adb con el comando:

```
$ adb reboot recovery
```

El recovery que hay instalado es el *Team Win Recovery Project 3.0 (TWRP)*. Una vez en el menú principal, hay que ir a *Wipe ->Advanced Wipe*. Aquí seleccionamos la partición */data* y marcamos la opción *Change or repair file system ->Change file system*. Escogemos el formato EXT4 y realizamos el wipe. Tras ello, se reiniciará el terminal y habrá que

Benchmark / Formato	EXT4	F2FS
RL Bench	21,4 s	14,4 s
CF Bench	33478 puntos	34172 puntos
0xBenchmark	1147 puntos	1168 puntos
AnTuTu Benchmark	30134 puntos	31306 puntos
Quadrant Benchmark	9588 puntos	10736 puntos

Tabla 8.1: Resultados de ejecución de todos los benchmark usando EXT4 y F2FS.

realizar los mismos pasos para la partición */cache*.

En la Tabla 8.1 se muestran los resultados de ambos sistemas de ficheros (EXT4 y F2FS) de cada uno de los benchmarks. Como se puede observar en los resultados de la tabla, el sistema de ficheros F2FS supera en puntuación todas las pruebas de benchmark realizadas. Por tanto, se puede concluir que para dispositivos que usan tarjetas de memoria tipo NAND, un sistema de ficheros como F2FS puede ser más conveniente que otro como EXT4, que es más recomendado para entornos de escritorio.

## 8.7. Análisis valorativo

Finalmente, se ha realizado un análisis del tiempo de dedicación y dificultad a modo orientativo de cada una de las prácticas. En la Tabla 8.2 se muestra una ponderación de las horas aproximadas que habría que dedicar a cada práctica sugerida así como el nivel de dificultad que supone su realización. El curso recomendado indica en qué curso de la universidad debería realizarse la práctica en relación a su nivel de dificultad.

Práctica	Dedicación (h)	Difucultad	Curso recomendado
1	4	Media	2º, 3º
2	2	Alta	4º
3	2	Media	2º, 3º
4	2	Media	2º, 3º
5	2	Baja	2º

Tabla 8.2: Número de horas de dedicación, dificultad de cada práctica de laboratorio y curso recomendado.

# Capítulo 9

## Sostenibilidad y compromiso social

### 9.1. **Ámbito económico**

Para estudiar la viabilidad del proyecto, debe realizarse una evaluación de los costes que su desarrollo conlleva, así como estudiar los recursos de los que se disponen y gestionarlos adecuadamente. Para ello, en el capítulo 5 se explica detalladamente a parte de todos los costes, los posibles imprevistos y desviaciones del presupuesto y cómo hacerles frente.

Cabe destacar que todos los recursos de software son gratuitos y los recursos de hardware aunque no tienen coste nulo, fueron adquiridos previamente y para un uso distinto, de manera que su coste de uso es inferior a su coste de adquisición. Por otro lado, como se comenta en el apartado 5.2.5 de Presupuesto, el producto final no espera comercializarse, sino que es para un uso académico, por tanto, no se pretende obtener ningún beneficio económico de él.

Tomando todos estos detalles en cuenta, el proyecto se valora con un 9 en sostenibilidad económica, ya que todos los recursos no humanos han tenido coste nulo.

### 9.2. **Ámbito social**

Existe un gran beneficio social que se puede aprovechar del producto final. Si la propuesta del laboratorio virtual se ve factible, puede llegar a ser usada en la propia asignatura de Sistemas Operativos, así que se podrían beneficiar tanto los estudiantes como los profesores que la imparten. Además, en caso de un mayor éxito del previsto, podría incluso ser usado por otras universidades y centros académicos, por lo cual, el beneficio social abarcaría aún más gente todavía.

Debido a ello, se valora la sostenibilidad social con 8. No obtiene la máxima puntuación ya que hay un riesgo de que el producto final no llegue a ponerse en práctica. Al fin y al cabo, se trata de una propuesta.

### 9.3. **Ámbito ambiental**

Considerando que tanto el ordenador portátil como el teléfono móvil fueron adquiridos con anterioridad a la realización del proyecto, su uso no ha supuesto un problema ambiental. Además, el portátil es usado con otros fines por lo que su aprovechamiento va más allá del proyecto en sí.

En cuanto al software usado, una parte importante está en la nube y en servicios de Internet, por ejemplo: Google Drive, ShareLaTeX, Gmail, etc. Esto implica que el consumo de la CPU es menor, ya que el acceso a estos recursos es vía navegador, y su consumo es menor a que si hubiera una aplicación copia de cada servicio funcionando en local. Por otro lado, el producto final es software, por lo que no se requiere de maquinaria de fabricación y solo hay consumo energético del ordenador y el teléfono móvil.

En caso que el producto final llegara a usarse, se acabaría usando en los ordenadores del laboratorio. Esto supone un gasto energético, pero como en el caso de que el producto final no se usara aún así se seguirían usando estos ordenadores para otras prácticas de laboratorio, este consumo energético es inevitable y no aumentaría debido a este proyecto. Debido a todos estos detalles, su puntuación en cuanto a sostenibilidad ambiental es de 9.

### 9.4. **Matriz de sostenibilidad**

Tras estudiar los tres ámbitos de sostenibilidad, se puede definir una matriz como la de la Tabla 9.1.

<b>Sostenibilidad</b>	<b>Económica</b>	<b>Social</b>	<b>Ambiental</b>	<b>Total</b>
Planificación	Viabilidad económica	Mejora de la calidad de vida	Análisis de recursos	
Valoración	9	8	9	<b>26</b>

Tabla 9.1: Matriz de sostenibilidad

# Capítulo 10

## Conclusiones

Desde el surgimiento de Android así como de otros sistemas operativos embebidos, se ha abierto la puerta a un mundo de investigación y desarrollo de sistemas móviles. Ya sea en hardware o en software, se han realizado grandes avances en este campo. No obstante, el estado del arte muestra que en el mundo de la enseñanza, el estudio de Android suele basarse mayoritariamente en desarrollar aplicaciones para él, dejando de lado su estudio como sistema operativo. En este proyecto, se ha pretendido implementar un laboratorio virtual que reúna todas las herramientas necesarias para poder ser capaz de trabajar cómodamente con el código fuente de Android y ayudar al desarrollador a testarlo tanto en un dispositivo físico como en un emulador.

El reto más grande de este proyecto ha sido el de analizar la mejor forma de mejorar la asignatura de Sistemas Operativos enfocándola al estudio de Android como sistema operativo. Para ello, se han analizado las distintas herramientas que deberían estar incluidas en el laboratorio virtual, así como una evaluación objetiva de los distintos emuladores disponibles para decidir cuál incluir en el laboratorio. Se han sugerido además tres maneras distintas de implementación y, finalmente, se han propuesto e implementado una serie de prácticas de laboratorio con el objetivo de aportar una serie de tareas prácticas para que el estudiante pueda hacer un uso del laboratorio virtual al mismo tiempo que aprende a trabajar en Android.

El código fuente de Android empleado para este proyecto no es la rama principal del proyecto AOSP sino la versión de *CyanogenMod*. El motivo es que el dispositivo móvil empleado para la realización de este proyecto (*Motorola LTE E*) solo es compatible con este último. No obstante, el laboratorio virtual puede ser usado con cualquier otra distribución del código fuente de Android, al igual que con cualquier otro terminal móvil basado en Android, por lo que la ROM que escojamos no debería de ser un problema siempre y cuando



ésta sea compatible con el dispositivo final. Cabe destacar como curiosidad que durante la realización de este proyecto (concretamente el día 31 de diciembre de 2016), la empresa *Cyanogen Inc.* encargada de su ROM *CyanogenMod* decidió cesar su actividad debido a la baja rentabilidad de su producto. No obstante, el código fuente sigue estando disponible para toda la comunidad de desarrolladores. A parte de *CyanogenMod*, existen muchas otras alternativas como por ejemplo *Paranoid Android* [50], *OmniRom* [51] o *AOKP* [52] .

El proyecto no ha sufrido ninguna desviación temporal respecto a la planificación inicial. Las reuniones con el director han ayudado a llevar un control semanal del progreso del proyecto. En cuanto a la parte económica, tampoco ha habido ninguna desviación ya que se han cumplido con las fechas marcadas de entrega y el material empleado para el proyecto ha sido el propuesto inicialmente.

Desde el punto de vista subjetivo del autor, el proyecto en sí ha sido un gran reto debido a que al inicio ni el director ni el autor tenían una idea totalmente definida de cómo sería el producto final y fue durante las primeras semanas que fueron definiéndose las distintas funcionalidades y herramientas que irían incluidas. Algunos aspectos del proyecto han sido reconsiderados y modificados respecto al planteamiento inicial, aún así, creo que finalmente se han cumplido los objetivos marcados inicialmente y con todo ello, los estudiantes de Sistemas Operativos deberían de ser capaces de usar este laboratorio así como llevar a cabo cada una de las prácticas propuestas en él.

# Bibliografía

- [1] GNU/Linux - Wikipedia, la enciclopedia libre [en línea]  
<https://es.wikipedia.org/wiki/GNU/Linux>  
consultado el 20/10/2016 a las 6:42
  
- [2] Android - Wikipedia, la enciclopedia libre [en línea]  
<https://es.wikipedia.org/wiki/Android>  
consultado el 20/10/2016 a las 6:46
  
- [3] 2 Billion Consumers Worldwide to Get Smart(phones) by 2016 eMarketer Newsroom [en línea]  
<http://www.emarketer.com/newsroom/index.php/emarketer-2-billion-consumers-worldwide-smartphones-2016>  
consultado el 20/09/2016 a las 19:21
  
- [4] Crear y administrar dispositivos virtuales | Android Studio [en línea]  
<https://developer.android.com/studio/run/managing-avds.html?hl=es-419>  
consultado el 19/10/2016 a las 18:05
  
- [5] Teaching Operating Systems Using Android, Dept. of Computer Science - Columbia University [en línea]  
<http://systems.cs.columbia.edu/files/wpid-sigcse2012-android.pdf>  
consultado el 20/09/2016 a las 21:10
  
- [6] Sistema embebido - Wikipedia, la enciclopedia libre [en línea]  
[https://es.wikipedia.org/wiki/Sistema\\_embebido](https://es.wikipedia.org/wiki/Sistema_embebido)  
consultado el 19/10/2016 a las 19:40
  
- [7] TA New Curriculum for Teaching Embedded Systems at the University of Ljubljana [en línea]  
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.131.688&rep=rep1&type=pdf>  
consultado el 19/10/2016 a las 18:27

- 
- [8] Design and Development of a Web-based Interactive Software Tool for Teaching Operating Systems [en línea]  
[https://www.learntechlib.org/p/111509/article\\_111509.pdf](https://www.learntechlib.org/p/111509/article_111509.pdf)  
consultado el 19/10/2016 a las 18:54
- [9] Teaching Operating Systems – Windows Kernel Projects [en línea]  
[http://ims.mii.lt/ims/konferenciju\\_medziaga/SIGCSE'10/docs/p490.pdf](http://ims.mii.lt/ims/konferenciju_medziaga/SIGCSE'10/docs/p490.pdf)  
consultado el 19/10/2016 a las 19:50
- [10] The leading OS for PC, tablet, phone and cloud | Ubuntu [en línea]  
<https://www.ubuntu.com/>  
consultado el 21/10/2016 a las 10:12
- [11] What is Scrum? An Agile Framework for Completing Complex Projects - Scrum Alliance [en línea]  
<https://www.scrumalliance.org/why-scrum>  
consultado el 25/09/2016 a las 17:45
- [12] LaTeX - A document preparation system [en línea]  
<https://www.latex-project.org/>  
consultado el 25/09/2016 a las 18:11
- [13] About Us - ShareLaTeX, Editor de LaTeX online [en línea]  
<https://es.sharelatex.com/about>  
consultado el 25/09/2016 a las 18:13
- [14] Estudios de Remuneración | Page Personnel [en línea]  
<http://www.pagepersonnel.es/sites/pagepersonnel.es/files>  
consultado el 07/10/2016 a las 10:11
- [15] Android - Wikipedia, la enciclopedia libre [en línea]  
[https://es.wikipedia.org/wiki/Android#Usos\\_y\\_dispositivos](https://es.wikipedia.org/wiki/Android#Usos_y_dispositivos)  
consultado el 10/11/2016 a las 20:28
- [16] Blink - The Chromium Projects [en línea]  
<https://www.chromium.org/blink>  
consultado el 10/11/2016 a las 20:44
- [17] Khronos Releases OpenGL ES 3.0 Specification [en línea]  
<https://www.khronos.org/news/press/khronos-releases-opengl-es-3.0-specification>  
consultado el 10/11/2016 a las 20:51

- [18] Android Open Source Project [en línea]  
<https://source.android.com/>  
consultado el 10/11/2016 a las 21:25
- [19] Embedded Linux Experts - Free Electrons [en línea]  
<http://free-electrons.com/doc/training/android/android-slides.pdf/>  
consultado el 12/11/2016 a las 13:02
- [20] The Android boot process from power on [en línea]  
<http://www.androidenea.com/2009/06/android-boot-process-from-power-on.html>  
consultado el 12/11/2016 a las 16:51
- [21] Todo sobre las particiones de Android [en línea]  
<https://hipertextual.com/archivo/2014/01/particiones-android/>  
consultado el 15/11/2016 a las 20:12
- [22] Ext4 Howto - Ext4 [en línea]  
[https://ext4.wiki.kernel.org/index.php/Ext4\\_Howto#EXT4\\_features](https://ext4.wiki.kernel.org/index.php/Ext4_Howto#EXT4_features)  
consultado el 15/11/2016 a las 21:03
- [23] What is a journaling filesystem? – definition by The Linux Information Project [en línea]  
[http://www.linfo.org/journaling\\_filesystem.html](http://www.linfo.org/journaling_filesystem.html)  
consultado el 15/11/2016 a las 21:13
- [24] Sistema de archivos ext4 | NIS [en línea]  
<http://www.i-nis.com.ar/tutoriales/ext4>  
consultado el 15/11/2016 a las 21:07
- [25] F2FS - Wikipedia [en línea]  
<https://en.wikipedia.org/wiki/F2FS>  
consultado el 15/11/2016 a las 21:20
- [26] Requirements | Android Open Source Project [en línea]  
<https://source.android.com/index.html>  
consultado el 21/10/2016 a las 11:15
- [27] How To Build CyanogenMod For Motorola Moto E 2015 LTE ("surnia") - Cyanogen-Mod [en línea]  
[https://wiki.cyanogenmod.org/w/Build\\_for\\_surnia](https://wiki.cyanogenmod.org/w/Build_for_surnia)  
consultado el 21/10/2016 a las 11:17

- [28] Moto E (2nd Gen.) | Motorola [en línea]  
<http://www.motorola.es/products/moto-e-gen-2>  
consultado el 30/10/2016 a las 11:33
- [29] Manual de rooteo para el Motorola Moto E 2015 [en línea]  
<http://www.movilzona.es/2015/04/29/motorola-moto-e-2015-manual-root/>  
consultado el 30/10/2016 a las 11:44
- [30] Information: Motorola Moto E 2015 LTE ("surnia") - CyanogenMod [en línea]  
[https://wiki.cyanogenmod.org/w/Surnia\\_Info](https://wiki.cyanogenmod.org/w/Surnia_Info)  
consultado el 30/10/2016 a las 12:26
- [31] Ejecutar apps en el emulador de Android | Android Studio [en línea]  
<https://developer.android.com/studio/run/emulator.html?hl=es-419>  
consultado el 29/10/2016 a las 11:31
- [32] Features – Genymotion Android Emulator [en línea]  
<https://www.genymotion.com/features/>  
consultado el 29/10/2016 a las 11:41
- [33] Bluestacks Android Emulador para PC y Mac [en línea]  
<http://www.bluestacks.com/es/about-us/app-player.html>  
consultado el 29/10/2016 a las 11:47
- [34] Android-x86 - Porting Android to x86 [en línea]  
<http://www.android-x86.org/>  
consultado el 29/10/2016 a las 12:15
- [35] YouWave, A world for Android on PC [en línea]  
<https://youwave.com/>  
consultado el 29/10/2016 a las 12:24
- [36] QEMU [en línea]  
[http://wiki.qemu.org/Main\\_Page](http://wiki.qemu.org/Main_Page)  
consultado el 29/10/2016 a las 16:05
- [37] Oracle VM VirtualBox [en línea]  
<https://www.virtualbox.org/>  
consultado el 29/10/2016 a las 14:28
- [38] AnTuTu Benchmark – Know Your Android Better [en línea]  
<http://www.antutu.com/en/index.shtml>  
consultado el 29/10/2016 a las 19:11

- [39] Pixel, Phone by Google [en línea]  
<https://www.android.com/phones/pixel/>  
consultado el 30/10/2016 a las 19:37
- [40] Codenames, Tags, and Build Numbers | Android Open Source Project [en línea]  
<https://source.android.com/source/build-numbers.html#source-code-tags-and-builds>  
consultado el 30/10/2016 a las 19:35
- [41] CyanogenMod · GitHub [en línea]  
<https://github.com/CyanogenMod/>  
consultado el 02/11/2016 a las 09:14
- [42] Developing | Android Open Source Project [en línea]  
<https://source.android.com/source/developing.html>  
consultado el 02/11/2016 a las 09:25
- [43] ccache - Overview [en línea]  
<https://ccache.samba.org/>  
consultado el 02/11/2016 a las 16:28
- [44] Android Build System - eLinux.org [en línea]  
[http://elinux.org/Android\\_Build\\_System#Overview](http://elinux.org/Android_Build_System#Overview)  
consultado el 02/11/2016 a las 10:12
- [45] make(1) - Linux man page [en línea]  
<https://linux.die.net/man/1/make>  
consultado el 04/11/2016 a las 14:05
- [46] CyanogenMod Downloads [en línea]  
[https://download.cyanogenmod.org/?device=flounder\\_lte](https://download.cyanogenmod.org/?device=flounder_lte)  
consultado el 02/01/2016 a las 10:47
- [47] Clonezilla live [en línea]  
<http://clonezilla.org/clonezilla-live.php>  
consultado el 05/11/2016 a las 11:16
- [48] GitHub - libfuse/sshfs: A network filesystem client to connect to SSH servers [en línea]  
<https://github.com/libfuse/sshfs>  
consultado el 06/11/2016 a las 13:32
- [49] What is Samba? [en línea]  
[https://www.samba.org/samba/what\\_is\\_samba.html](https://www.samba.org/samba/what_is_samba.html)  
consultado el 06/11/2016 a las 13:34

- [50] Paranoid Android - Oficial [en línea]  
<http://paranoidandroid.co/>  
consultado el 13/07/2017 a las 11:01
- [51] OmniROM [en línea]  
<https://omnirom.org/>  
consultado el 13/07/2017 a las 11:02
- [52] AOKP.co [en línea]  
<http://aokp.co/>  
consultado el 13/07/2017 a las 11:03