ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"EXTENSIÓN DE CAPACIDADES DE TRANSMISIÓN DE DATOS EN SMARTPHONES CON SISTEMA OPERATIVO ANDROID"

Informe de Materia de Graduación

Previo a la obtención del Título de:

INGENIERO EN CIENCIAS COMPUTACIONALES ESPECIALIZACIÓN SISTEMAS MULTIMEDIA

INGENIERO EN CIENCIAS COMPUTACIONALES ESPECIALIZACIÓN SISTEMAS TECNOLÓGICOS

Presentado por:

Johnny Omar Ramirez Malavé
Washington Andrés Sánchez Alvarez

GUAYAQUIL – ECUADOR

Año: 2013

AGRADECIMIENTO

Al todopoderoso, Dios por siempre levantarme en el momento que estoy caído, y por demostrarme que cada experiencia en la vida es lo que forma tu carácter y que depende de cada uno de nosotros saber cómo aprovecharlo.

A mami Luisa y a papi Gerardo, los pilares de mi vida, que sin ellos no sería lo que soy si no hubieran realizado todos los sacrificios que han hecho por mí, no estaría en este punto de mi vida.

A mi esposa Isabel, que sin sus insistencias no me hubiera dedicado a terminar la tesis y concluir mi carrera, nuestra familia se completara pronto y sé que al igual que mis padres, haremos los sacrificios necesarios para que nuestro hijos sean mejores que nosotros.

A mi amigo Johnny, por sacrificar horas de sueño y tiempo con su familia para trabajar en nuestra tesis, el esfuerzo valió la pena.

Al Dr. Ochoa, que sin su ayuda no hubiéramos podido terminar esta tesis.

Washington Andres Sanchez Alvarez

AGRADECIMIENTO

Primeramente agradezco a Dios que es el me acompaña en cada momento de mi vida, a mis padres que siempre estuvieron en todo momento apoyándome y aun lo siguen haciendo en esta etapa de mi vida.

A mi esposa y a mi hijo que son la razón por la cual me encuentro donde estoy y aspiro cosas mucho mejores.

A mi compañero de tesis que muchas veces tuvimos que sacrificar trabajo y noches de sueño para cumplir con nuestros objetivos.

Al Dr. Daniel Ochoa que sin su ayuda y su paciencia no hubiéramos terminado nuestra tesina.

A todos mis profesores que mi brindaron sus conocimientos durante toda mi vida universitaria.

Johnny Omar Ramirez Malavé

DEDICATORIA

A Dios, por enseñarme que todo nos fortalece.

A mi madre, que nunca podré terminar de recompensar todo lo que ha hecho por nosotros.

A mi padre, que sacrificó su salud para que nunca nos falte nada, siempre estaré allí para cuidarte mi viejo.

A mis hermanos Anita y Jerome, a pesar de las distancias siempre cuentan conmigo.

A mi hermana Desiree, quiero que tomes este logro como una inspiración para tu futuro.

A Isabel, la mujer que amo, mi esposa y la madre de mis hijos, que sin su apoyo no podría haber terminado los de la tesis.

Washington Andrés Sánchez Alvarez

DEDICATORIA

A mi madre Mabel Malavé que me ayudo siempre con mis estudios desde la escuela donde aprendí mi primera letra hasta el este momento.

A mi padre que con su ayuda y sus enseñanzas llegue a la universidad.

A mi esposa Liliana Mina y a mi hijo Johnny Ramírez que son mi ayuda y las razones por las cuales he salido adelante.

Johnny Omar Ramirez Malavé

TRIBUNAL DE SUSTENTACIÓN

Dr. Daniel Ochoa Donoso

PROFESOR DE LA MATERIA DE GRADUACIÓN

MSc. Patricia Chávez Burbano

PROFESORA DELEGADA POR LA UNIDAD ACADEMICA

DECLARACIÓN EXPRESA

“La responsabilidad del contenido de este Informe, nos corresponde exclusivamente; y el patrimonio intelectual de la misma a la Escuela Superior Politécnica del Litoral”.

Johnny Omar Ramirez Malavé

Washington Andrés Sánchez Alvarez

RESUMEN

El siguiente trabajo demuestra una de las principales ventajas de usar el sistema operativo Android: el poder manipularlo para explotar plataformas móviles. En nuestro proyecto habilitamos el módulo del núcleo de Android necesario para utilizar un transmisor de radio y así poder recibir datos desde puntos de difícil acceso o carentes de acceso a la red celular.

En este proyecto nos enfocamos en optimizar la velocidad de obtención de datos, haciendo uso de librerías y marcos de trabajo para el núcleo de Android y el núcleo de Linux.

En nuestros experimentos se analizó el rendimiento de una aplicación desarrollada para el sistema operativo Android, en base al número de paquetes perdidos durante la trasmisión y la distancia a la que fueron enviados. Este trabajo permite usar, en principio, un teléfono como un nodo de una red de sensores.

ÍNDICE GENERAL

RESUMEN VIII

ÍNDICE GENERAL IX

ÍNDICE DE FIGURAS XIII

ÍNDICE DE TABLAS XIV

GLOSARIO XV

ABREVIATURAS XVII

INTRODUCCIÓN XX

CAPÍTULO 1: 1

1.1 Objetivos 1

1.1.1 Objetivo General 1

1.1.2 Objetivos Específicos 2

1.2 Definición del Problema 2

1.3 Antecedentes 3

1.4 Justificación 6

1.5 Metodología 6

1.6 Limitaciones 7

1.7 Equipos Utilizados 8

CAPÍTULO 2: 9

MARCO TEORICO 9

2.1 Android 9

2.2 Ventajas sobre otras plataformas 12

2.3 Máquina Virtual Dalvik 14

2.4 Desarrollo de Aplicaciones en Android 21

2.4.1 Kit de Desarrollo de Software Android (SDK) 21

2.4.2 Kit de Desarrollo Nativo Android (NDK) 22

2.4.3 Puente de depuración Android (ADB) 24

CAPITULO 3: 25

ARQUITECTURA DE ANDROID 25

3.1 Núcleo de Linux 26

3.2 Librerías 29

3.3 Tiempo de Ejecución 32

3.4 Marco de Trabajo de aplicaciones 32

3.5 Capa de Aplicaciones 34

CAPÍTULO 4: 36

DISPOSITIVOS FUTUROS DE TECNOLOGÍA INTERNACIONAL Y DESARROLLO DE LA APLICACIÓN ANDROID 36

4.1 Dispositivos Futuros de Tecnología Internacional (FTDI) 36
4.1.1 Integración con Android 37
4.1.2 Soporte USB OTG (USB Anfitrión) 39
4.2 Habilitar módulo FTDI en Android 40
4.2.1 Código fuente del Núcleo 41
4.2.2 Modo Recuperación y Superusuario 42
4.2.3 Configuración del Núcleo 44
4.2.4 Compilación del Núcleo 47
4.2.5 Instalación del módulo FTDI en Android 49
4.2.6 Errores al instalar Módulos 51
4.3 Diseño de la Aplicación 52
4.3.1 Primer Diseño 53
4.3.2 Segundo Diseño 54
4.4 Implementación 58
4.4.1 Permisos para el Dispositivo 58
4.4.2 Interfaz de comunicación entre Java y C 59
4.4.3 Funcionamiento de la Aplicación 61
4.5 Experimento 64
4.6 Conclusiones 66
CONCLUSIONES Y RECOMENDACIONES 67
ANEXOS 71

BIBLIOGRAFIA 90

ÍNDICE DE FIGURAS

Figura 2.1.- Funcionamiento de la herramienta dx [11] 17

Figura 2.2.- Comparación entre archivos Class y Dex [13] 19

Figura 2.3.- JNI, el puente entre Java y Código Nativo 23

Figura 3.1.- Arquitectura de Android [15] 26

Figura 4.1.- Comunicación del teléfono mediante el FTDI 38

Figura 4.2.- Diagrama de Bloques del chip FT312D 39

Figura 4.3.- Pasos para habilitar módulo FTDI en android 41

Figura 4.4.- Pantalla del Modo Recuperación 43

Figura 4.5.- Aplicación Superusuario 44

Figura 4.6.- Menú de configuración 46

Figura 4.7.- Controlador Simple de Puerto Serial FTDI 47

Figura 4.8.- Instalar y listar módulos del núcleo 51

Figura 4.9.- Primer Diseño de la Aplicación 54

Figura 4.10.- Segundo Diseño de la Aplicación 57

Figura 4.11.- Compilación de la librería nativa 61

Figura 4.12.- Funcionamiento de la Aplicación 63

Figura 4.13.- Perdida de paquetes a diferentes frecuencias 65

ÍNDICE DE TABLAS

Tabla I.- Mercado de Sistemas operativos móviles a nivel mundial [5].....11

Tabla II.- Comparación entre Android, iOS y Windows Phone 8 [3]..... 13

Tabla III.- Comparación espacio de memoria entre Jar y Apk [6] 21

GLOSARIO

Complemento: Es un complemento, conector o extensión de una aplicación que usa para relacionarse con otra y aportarle una nueva función que generalmente es muy específica.

ClockworkMod: Es un modo de recuperación avanzado, este puede realizar más funciones que el modo recuperación de fábrica, pude instalar Custom Roms, Temas, etc. En la ciertos de Móviles se puede entrar presionando TECLA INICIO + ENCENDER + VOLUMEN ARRIBA.

Custom ROM: Estas Roms son modificadas por desarrolladores particulares y no dependen de la compañía o fabricantes.

Dalvik Cache: Es la aplicación que se encarga de almacenar el código de byte que se genera la Máquina Virtual Dalvik al iniciar el sistema operativo al cargarse las aplicaciones.

Makefile: Archivo que contiene la secuencia de instrucciones que se ejecutan al compilar el núcleo.

QEMU: Quick Emulator o Emulador rápido, es un producto de software

gratuito y código abierto que permite virtualizar hardware.

ROM Oficial: Es la Rom instalada desde Fabrica, su configuración depende de la Compañía Telefónica, también es llamada Rom Stock o Rom Original.

Root: O superusuario, es tener todos los permisos de acceso en un sistema operativo.

Widgets: Aplicación que te muestra información actual sin navegar por distintas páginas.

Wipe Cache Partition: Es la opción que permite borrar todos los datos almacenados en la memoria cache es decir los datos de navegación y de aplicaciones que se encuentren en cache.

Wipe Dalvik Cache: Limpia la memoria Dalvik Cache.

Wipe Data: Borra todos los datos almacenados en el Móvil como contactos, aplicaciones, notas, mensajes, dejándolo con la configuración de fábrica.

ABREVIATURAS

AAC: Codificación de Audio Avanzado

ADB: Puente de Depuración de Android

ADT: Herramientas de Desarrollo de Android

AMR: Multi-tasa adaptativo

API: Interfaz de Programación de la Aplicación

APK: Paquete de Archivo de la Aplicación

ARM: Maquina avanzada de conjunto de instrucciones reducidas

AVC: Codificación de Video Avanzado

AVD: Dispositivo Virtual Android

CPU: Unidad Central de Procesamiento

CRC: Comprobación de Redundancia Cíclica

CTS: Limpio para enviar

DVM: Máquina Virtual Dalvik

EDGE: Tasas de Datos Mejoradas para la evolución de GSM

FIFO: Primero en Entrar, Primero en Salir

FTDI: Dispositivos de Tecnología Futura Internacional

GIF: Formato de Intercambio de Gráficos

GPRS: Servicio general de paquetes vía radio

GPS: Sistema de Posicionamiento Global

GSM: Sistema global para las comunicaciones móviles

HSCSD: Circuito conmutado de datos de alta velocidad

IDE: Entorno de Desarrollo Integrado

IEEE: Instituto de Ingenieros Eléctricos y Electrónicos

iOS: Sistema Operativo de iPhone

JVM: Máquina Virtual Java

JSE: Edición Estándar de Java

JME: Edición Móvil de Java

JAR: Archivo Java

JPG: Grupo Unión de Expertos Fotográficos

LED: Diodo Emisor de Luz

MMS: Servicio de Mensajería Multimedia

MPEG4: Grupo de Expertos de Imágenes con Movimiento)

NDK: Kit de Desarrollo Nativo

NFC: Comunicación de Campo Cercano

NMEA: Asociación Nacional de Electrónica Marina

OOM: Sin memoria

OpenGL: Librería Grafica Abierta

OTG: On the go

PNG: Red de Gráficos Portables

RAM: Memoria de acceso aleatorio

RTS: Solicitud para enviar

RXD: Dato recibido

SDK: Kit de Desarrollo de Software

SGL: Librería de Escenario Grafico

SL: Lenguaje de Sombreado

SMS: Servicio de Mensaje Corto

SSL: Capa de conexión segura

TXD: Dato transmitido

UART: Transmisor/Receptor Universal Asíncrono

USB: Bus Serial Universal

USBDM: USB Datos Menos

USBDP: USB Datos Más

VM: Máquina Virtual

WECA: Alianza de compatibilidad de redes inalámbricas Ethernet

YAFFS2: Sólo otro sistema de archivos flash

2D: 2 Dimensiones

3D: 3 Dimensiones

INTRODUCCIÓN

El uso de tabletas y teléfonos inteligentes en la vida diaria es una realidad palpable. Hoy en día los dos más grandes Sistemas Operativos para teléfonos inteligentes son iOS y Android, pero podemos remarcar que Android no solo es usado en estos teléfonos inteligentes o tabletas, sino que también es utilizado en dispositivos como relojes, cámaras fotográficas y hasta televisores.

La disminución de costos, la facilidad, gratuidad de las herramientas para programar, y la comercialización de teléfonos, con Android como Sistema Operativo, por parte de grandes compañías de telecomunicaciones, abrió un nicho de mercado que se encuentra en constante crecimiento para los desarrolladores de software de aplicaciones sencillas.

En el presente documento vamos a ir más allá de desarrollar una simple aplicación, se muestra como extender la funcionalidad de Android para que soporte dispositivos externos. En nuestro proyecto habilitamos los puertos de comunicación del teléfono y el núcleo del sistema operativo para utilizar una radio que recibe datos extraídos de un GPS remoto.

En el primer capítulo se define el objetivo de esta tesis, y se explica porque se decidió desarrollar el programa sobre el sistema operativo Android. Los

capítulos 2 y 3 explican la historia de Android y las ventajas que tiene sobre otros sistemas operativos para dispositivos móviles. En el capítulo 4 se explican los chips FTDI y como habilitar un módulo del núcleo de Android para poder comunicarnos con un dispositivo externo basado en FTDI, se listan los pasos a seguir, y se explica el diseño de la aplicación y sus diferentes versiones.

Finalmente, presentaremos datos estadísticos acerca de la eficiencia de ejecutar la aplicación desarrollada sobre un teléfono, y como se puede aprovechar la facilidad que presenta el teléfono para trasladarse a puntos de difícil acceso.

CAPÍTULO 1:
FORMULACIÓN DEL PROBLEMA

En esta sección se definen los objetivos de este trabajo, se plantea el problema a resolver, también incluiremos algunos antecedentes, justificación, metodología y limitaciones del trabajo y finalizaremos mencionando los equipos utilizados para el experimento.

1.1 Objetivos

Se dividen en un objetivo general donde se explica el propósito del trabajo de una manera amplia y los objetivos específicos en donde se explicara más explícitamente nuestro propósito.

1.1.1 Objetivo General

Comprobar que el teléfono inteligente con sistema operativo Android es un dispositivo con las mismas prestaciones que un computador de escritorio y que, al cumplir las características de un

software de código abierto, puede ser modificado para poder conectar dispositivos pocos comunes y así aprovechar su facilidad de transportación para proyectos en los que se necesite tomar datos en lugares de difícil acceso.

1.1.2 Objetivos Específicos

- Extender el núcleo de Android con la finalidad de habilitar el módulo FTDI en el teléfono y tener la capacidad de enviar y recibir datos por medio de un puerto serial.

- Desarrollar una aplicación en Android con la ayuda del NDK, y utilizar código nativo para la comunicación entre el teléfono y una radio con interfaz usb, que recibe coordenadas geográficas, las transfiere al teléfono y el teléfono grafica dichas coordenadas en un mapa.

1.2 Definición del Problema

Los equipos utilizados para tomar datos climáticos o meteorológicos, son ubicados en lugares particulares, remotos y, en la mayoría de los casos, de difícil acceso, por ejemplo en lo alto de una montaña, en la cercanía de un volcán, o en lo profundo de la selva. Teniendo en cuenta estos factores, la toma de datos en el mismo sitio donde se encuentran estos

dispositivos dificulta el trabajo y conlleva demasiado tiempo, por lo que es preferible que estos dispositivos cuenten con un sistema de conexión inalámbrica de tal manera que la descarga de información sea sencilla y rápida.

Las tecnologías inalámbricas que se pueden utilizar para realizar la conexión inalámbrica en estos lugares remotos pueden tener sus limitantes: la falta de cobertura de red celular, el rango de alcance de una red WIFI y bluetooth, los costos de comunicación vía satélite, etc. A diferencia de la gran cobertura y un relativo bajo costo que una radio puede tener.

Como se mencionó inicialmente, la ubicación de estos dispositivos genera problemas debido a su difícil acceso, por lo que pensar en el uso de un computador portátil, para realizar la descarga de información, resulta algo poco probable. Es aquí donde el teléfono inteligente, con cada vez mejores capacidades de procesamiento y su facilidad de movilización, se presenta como la mejor opción para realizar la tarea de la toma de datos.

1.3 Antecedentes

Se puede decir que el primer paso para la aparición del internet fue la

comunicación inalámbrica. Las redes inalámbricas aparecen en las computadoras personales a principio de los 80, y las primeras transmisiones inalámbricas utilizaban transceptores de infrarrojo que es una forma especial de transmisión por radio, mediante un haz de luz en el espectro de frecuencia infrarroja se envía de un transmisor a un receptor a una distancia corta por ese motivo la comunicación infrarroja necesita línea vista entre los dispositivos que se comunican.

Las redes inalámbricas basadas en radio aparecen en los años 90 cuando la potencia de procesamiento de los chips llego a ser lo suficiente para transmitir y recibir datos a través de radio. Las redes empezaron a ser propietarias y por lo tanto muy costosas y el principal inconveniente era que no se podían comunicar entre diferentes marcas así que ha mediado de los 90 aparece un estándar llamado IEEE 802.11 para las comunicaciones inalámbricas. En Abril del 2000 una asociación llamada WECA que posteriormente paso a llamarse WI-FI Alliance certifica la interoperabilidad de los equipos mediante la norma IEEE 802.11b con la marca WIFI que tenía un alcance de hasta 100 metros y trabaja en una banda libre de 2,4 GHz.

En la parte de la tecnología celular las redes inalámbricas han tenido un gran desarrollo. La tecnología celular tuvo gran aceptación, por lo que a

los pocos años de implantarse se empezó a saturar el servicio, hubo la necesidad de desarrollar otras formas de acceso múltiple al canal y transformar los sistemas analógicos a digitales, para dar cabida a más usuarios [1].

La comunicación celular consta de varias generaciones, la primera generación (1G) apareció en 1979 esta red fue analógica y estrictamente para voz y no contaba con seguridad alguna. La segunda generación (2G) apareció en 1990 y fue una red totalmente digital y emplea protocolos de comunicación más sofisticados parecidos a los actuales. Después apareció la generación 2.5G que contaba con capacidades extendidas como GPRS que es una transmisión vía radio mediante la comunicación de paquetes, HSCSD que es una mejora a la transmisión GSM llegando a ser 6 veces superior a GSM, EDGE que funciona como puente entre la tecnología 2G y 3G. La tercera generación (3G) se destaca por tener la convergencia de voz y datos con acceso inalámbrico a internet, esto la hace apta para aplicaciones multimedia y altas transmisiones de datos. La cuarta generación (4G) están completamente basadas en el protocolo IP convirtiéndose en una red de redes. Otra comunicación con la que podría contar un teléfono es la comunicación bluetooth que contienen un transceptor que recibe y transmite a una frecuencia de 2.4 GHz y alcanza un máximo de 10 metros.

1.4 Justificación

El estudio de la solución al problema de la comunicación en lugares de difícil acceso es necesario para el beneficio de personas y proyectos que necesitan realizar la labor de tomar datos para su posterior análisis, y por ende utilizan sistemas de sensores de temperatura, presión, nivel de agua, etc. Estos lugares donde se obtiene la información para su posterior procesamiento pueden ser zonas peligrosas, por ejemplo en el caso de la temperatura los datos podrían venir de una zona volcánica, donde sería peligroso para el ser humano tomar los datos directamente, o en el caso de la lectura del nivel del agua, salinidad, contaminación, etc., sería complicado para el ser humano acceder hasta el lugar donde se colocó el sensor que registra la información. En estos casos lo conveniente sería que los datos se recepten mediante sensores que se comuniquen de manera inalámbrica a algún dispositivo externo, en nuestro caso un teléfono inteligente.

1.5 Metodología

Para obtener los resultados en este trabajo se usó el método de la experimentación científica, ya que se establecen parámetros iniciales, luego se alteran y se controlan estos parámetros conforme se repite el experimento para obtener los resultados esperados.

Las pruebas realizadas durante los experimentos intentan determinar la efectividad de las radios conectadas a un teléfono inteligente. El número de muestras (tramas de información enviadas) fue de 100, y se modificó el intervalo de tiempo de espera que existe entre cada transmisión de estas tramas, y la distancia entre la computadora que las transmite y el teléfono inteligente que las recepta, y se mide el número de tramas perdidas por parte del teléfono inteligente.

Estos cálculos nos permitieron comparar la eficiencia teórica y la eficiencia practica obtenida por nuestra cuenta, en términos del número de paquetes enviados, y el número de paquetes recibidos.

1.6 Limitaciones

Esta tesis intenta demostrar que la eficiencia de un programa desarrollado para una computadora personal puede ser igual a un programa desarrollado para un teléfono inteligente.

Se limita a habilitar los módulos necesarios para conectar las radios, no se modifica el núcleo de android permanentemente. Las pruebas fueron realizadas dentro de un domicilio, no se están realizando las comparaciones de los valores obtenidos en un lugar cubierto y en un lugar descubierto. Las radios que se utilizan para Las tramas son

transmitidas y recibidas en radios del mismo fabricante, mismo modelos y con configuraciones iguales.

1.7 Equipos Utilizados

Para el desarrollo de la aplicación android se utilizó una computadora portátil con las siguientes características: memoria de tres gigabytes, con un CPU Core 2 Duo de 2.4 GHz y disco duro de 160 gigabytes.

Para la modificación del núcleo de Android y la instalación de la aplicación se utilizó una Tableta Samsung Galaxy Tab 2 Gt-P3313.

Para la transmisión de datos se utilizaron 2 radios Xbee Pro S1 con un alcance en exteriores de 1.6 kilómetros y en interiores 90 metros.

CAPÍTULO 2:
MARCO TEORICO

En esta sección se definirá cual es el propósito de Android, se compara con diferentes sistemas operativos móviles presentes en el mercado, y finalmente se definirá que es la máquina virtual dalvik usada por Android.

2.1 Android

Android es un sistema operativo basado en Linux, es decir, un núcleo de sistema operativo gratuito y que puede ser modificado según las necesidades del usuario. En sus inicios fue pensado para teléfonos inteligentes tal como lo son iOS, Symbian y BlackBerry OS [2].

Debemos tener claro la definición de Linux: No se refiere a todo el software de aplicaciones y servicios, se refiere solo al núcleo. Es el software que gestiona las llamadas al sistema y las interrupciones [2], [3].

Google continua liberando el código de las diferentes versiones de Android bajo la licencia Apache, lo que permite al software ser modificado en base a las necesidades de quien lo requiera, y que pueda ser distribuido principalmente por los fabricantes de dispositivos, aunque también por operadores móviles y desarrolladores. A pesar de que la interface de usuario es diferente y las aplicaciones de Android tienen un enfoque diferente a las de los programas que usamos comúnmente en nuestras computadoras personales, Android sigue siendo solo una distribución de Linux [3], [4].

El núcleo de Android se basa en el núcleo de Linux versión 2.6, y a partir de la versión Ice Cream Sandwich, en el núcleo de Linux versión 3.x. Adicionalmente, está constituido de librerías y API basados en C y Java, pero diseñado primordialmente para dispositivos con pantallas táctiles como teléfonos inteligentes y tabletas [3].

Android posee una larga comunidad de desarrolladores que incrementan la funcionalidad de los dispositivos por medio de aplicaciones escritas, en su mayoría, en una versión personalizada del lenguaje de programación java, haciendo uso de herramientas como el Android SDK [3].

La compañía ComScore, que se encarga de llevar estadísticas del mundo digital, ha publicado en junio del 2013 sus estadísticas sobre los teléfonos inteligentes, colocando en primer lugar a Google Android como la plataforma número 1, con el 52% del mercado de sistemas operativos para móviles tal como lo muestra la tabla I [5].

Tabla I.- Mercado de Sistemas operativos móviles a nivel mundial [5]

	% de teléfonos		
	Febrero - 2013	**Mayo - 2013**	**Cambio Punto**
Total Teléfonos Inteligentes Subscritos	100.0%	100.0%	N/A
Android	51.7%	52.4%	0.7
Apple	38.9%	39.2%	0.3
BlackBerry	5.4%	4.8%	-0.6
Microsoft	3.2%	3.0%	-0.2
Symbian	0.5%	0.4%	-0.1

Android hoy en día se ha convertido en el sistema operativo más utilizado en los teléfonos inteligentes a nivel mundial, y por ende en la principal elección de compañías de tecnología que requieren de un sistema operativo de bajo costo, configurable, ligero y óptimo para dispositivos de alta tecnología sin tener que diseñar uno desde el inicio. Resultado de esto, a pesar de que principalmente fue diseñado para teléfonos y tabletas, ha sido usado en aplicaciones en televisores, consolas de juegos y otros dispositivos [3], [4].

2.2 Ventajas sobre otras plataformas

Actualmente existen varios Sistemas Operativos para dispositivos Móviles disponibles en el mercado: Symbian, IPhone, Windows Mobile, BlackBerry, Edición Móvil de Java, Linux Mobile, entre otros. A continuación mostramos una tabla comparativa entre los siguientes Sistemas Operativos con mayores expectativas en el mercado: iOS6, Android 4.X y Windows Phone 8.

Entre las principales ventajas que posee Android que podemos ver en la tabla II, tenemos las siguientes:

La tecnología Comunicación de Campo Cercano (NFC) es una tecnología de comunicación inalámbrica que posee un muy corto alcance (10 cm máximo) y una muy alta frecuencia que se usa para el paso de datos entre dispositivos. Android aprovecha esta tecnología para el intercambio prácticamente inmediato de grandes datos entre dos dispositivos, como música y videos [8].

El concepto de multitarea no es exclusivo de Android, iPhone y Windows Phone lo tienen también, pero con sus limitantes. Android nos permite ejecutar tantas tareas en paralelo como el hardware del teléfono lo

permita, a diferencia de los otros sistemas operativos que permiten solo a ciertos procesos su ejecución en paralelo [6], [7].

Tabla II.- Comparación entre Android, iOS y Windows Phone 8 [6]

S.O. Móvil	Android 4.1 Jelly Bean	iOS6	Windows Phone 8
Núcleo de Sistema Operativo	Linux	OS X	Windows NT
Código Abierto	Si	No	No
Multi-tarea	Si	Limitado	Limitado
Hardware Soportado	Una amplia variedad de dispositivos	iPhone, iPad, iPod touch	Una variedad de dispositivos
Soporte Flash	Si	No	No
Soporte Java	Si	No	No
Teclado	Físico y Virtual	Virtual	Físico y Virtual
Personalización Interface de Usuario	Si, nativo o con el uso de un sin número de aplicaciones	Limitada	Limitada
Reconocimiento de Voz	Si	Si	Si
Reconocimiento de Voz fuera de línea	Si	No	No
NFC	Si	No	Si
Automatización de tareas	Si	No	Si
Almacenamiento Extraíble	Si	No	Si
Soporte para Tabletas	Si	Si	No
Base de Datos integrada	Si	Si	Si

Una de las principales características a favor de Android, es el hecho de que se pueden conectar varios dispositivos (teclados, mouses, memorias portátiles usb, etc.) sin la necesidad de que sean dispositivos creados específicamente para dicho fabricante.

2.3 Máquina Virtual Dalvik

Al igual que en una computadora de escritorio, las aplicaciones deben poseer su propio ambiente sobre el cual puedan ejecutarse en Android, vamos a presentar una explicación de lo que es una máquina virtual y porque se decidió crear una específicamente para Android.

Para iniciar una máquina virtual es una aplicación que simula el funcionamiento de una computadora y puede ejecutar un sistema operativo como si se tratara de una computadora real. Una de las principales características que tienen las máquinas virtuales es que los procesos que pueden ejecutar son limitados por los recursos (memoria RAM, espacio en disco) asignados a estas, es decir, no pueden salir del espacio de memoria asignado a la misma [9].

Las máquinas virtuales pueden ser agrupadas en dos categorías, en base a su funcionalidad, estas categorías son:

Máquinas virtuales de sistema.- permiten a la máquina física multiplicarse entre varias máquinas virtuales, cada una ejecutando su propio sistema operativo [10].

Máquinas virtuales de proceso.- en este caso se ejecutan tal como lo hace un proceso cualquiera dentro de un sistema operativo y únicamente soporta un proceso. La máquina se inicia automáticamente en el momento que se va a ejecutar un proceso y se detiene automáticamente al momento de que se finaliza. Su objetivo principal es el de proporcionar un ambiente de ejecución que sea independiente tanto de la arquitectura como del sistema operativo, que se oculten estos detalles y permita que el programa se ejecute siempre de la misma manera sobre cualquier plataforma. El ejemplo más conocido es el de la máquina virtual de Java (JVM) [10]

Uno de los inconvenientes de las máquinas virtuales es que están limitados a los recursos que se le asignan, es decir, en base a los recursos que tiene disponible el sistema “anfitrión”, esto tiene como efecto el hacer lento al sistema, es decir, el programa no alcanzará la misma velocidad de ejecución que si lo instaláramos directamente en el sistema operativo "anfitrión" o si se ejecutara directamente sobre la plataforma de hardware [10].

Google seleccionó a Java como el lenguaje para el desarrollo de aplicaciones Android, pero a pesar de esto no utiliza la JVM para ejecutar sus aplicaciones, si no que creó su propia máquina virtual llamada Dalvik. Dalvik es la máquina virtual de procesos que se encarga de ejecutar aplicaciones en dispositivos que ejecutan el sistema operativo Android, cada aplicación ejecuta su propio proceso, y por ende, su propia instancia de la Máquina Virtual Dalvik [12].

La máquina virtual de Java utiliza el bytecode de java para ejecutar sus instrucciones. Este bytecode es un código intermedio más abstracto que el código de máquina, usualmente es el resultado de un compilador del lenguaje de programación Java, pero puede ser generado desde otros lenguajes. Aunque se utiliza a Java como lenguaje para programar las aplicaciones Android, el Java bytecode no puede ser ejecutado en el sistema operativo Android. De igual manera, las librerías que utiliza Android tienen pequeños cambios respecto a las utilizadas en JSE (Edición Estándar de Java) o en JME (Edición Móvil de Java).

El objetivo fundamental del Dalvik VM es el mismo que el de cualquier máquina virtual, permitir que el código sea compilado a un código que pueda ser ejecutado en cualquier máquina, o para el caso de Android, en cualquier teléfono inteligente o dispositivo con este sistema operativo.

Como se muestra en la figura 2.1 los programas son compilados a bytecode de java, luego un programa llamado dx (incluido en el SDK de Android) es usado para convertir los archivos de las clases en Java (.class) al formato Dex (Dalvik Ejecutable), un formato que fue diseñado para la optimización del almacenamiento y ejecución en el procesador [11], [14].

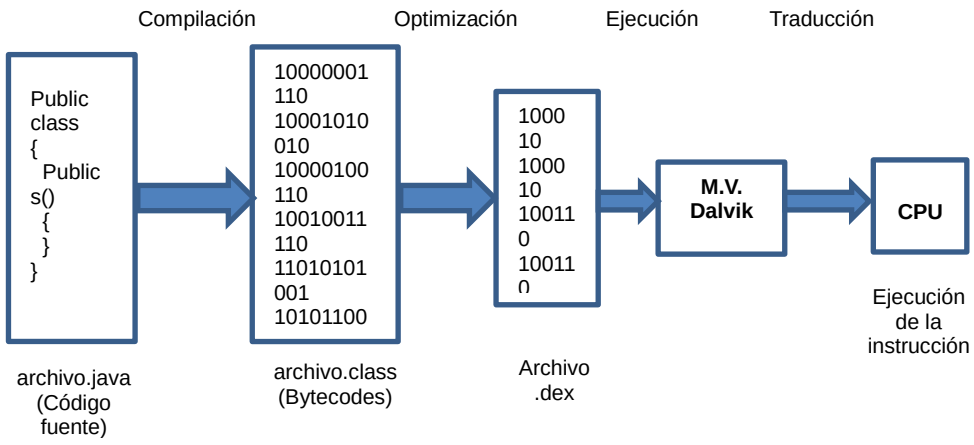

Figura 2.1.- Funcionamiento de la herramienta dx [11]

Una de las razones por las cuáles se decidió no utilizar la máquina virtual de Java es la necesidad de optimizar el uso de recursos en una plataforma dónde la memoria, procesador y almacenamiento son escasos. La ventaja que tiene Dalvik VM sobre la máquina virtual de Java es entendida con la arquitectura que manejan:

La máquina virtual de Java, que la mayoría de computadoras personales modernas tienen instalada, se basa en el uso de pilas. Las máquinas con arquitectura de pila deben usar instrucciones para cargar toda la información necesaria en la pila y entonces poder manipularla, no es posible acceder aleatoriamente por lo que se complica el poder generar código eficiente [11], [13].

De modo contrario, Dalvik se basa en el uso de registros, lo que automatiza la generación de código y la reutilización de comandos, haciendo el acceso a datos más rápido y veloz [13].

Otra característica importante de Dalvik VM es que fue diseñada para que múltiples instancias de esta se ejecuten de manera simultánea, lo esto considerando las limitaciones de memoria que existen en teléfono inteligentes, principalmente en teléfonos de gama baja. El objetivo de esto es evitar que cuando se presente algún fallo en una aplicación, esto afecte el correcto funcionamiento de otras. Entre las principales diferencias que hacen al Dalvik VM que el JVM para su uso en dispositivos móviles podemos indicar el hecho de que utiliza una pila de instrucciones de 16 bits, a diferencia del Java bytecode estándar, que usa solo 8 bits, que trabaja directamente sobre variables locales, lo cual junto al uso de un conjunto de constantes que usa 32 bytes, simplifica y

acelera su intérprete [13]. La figura 2.2 contrasta el formato usado por el archivo .class (usado por la JVM) con el formato usado por el archivo .dex (usado por Dalvik VM).

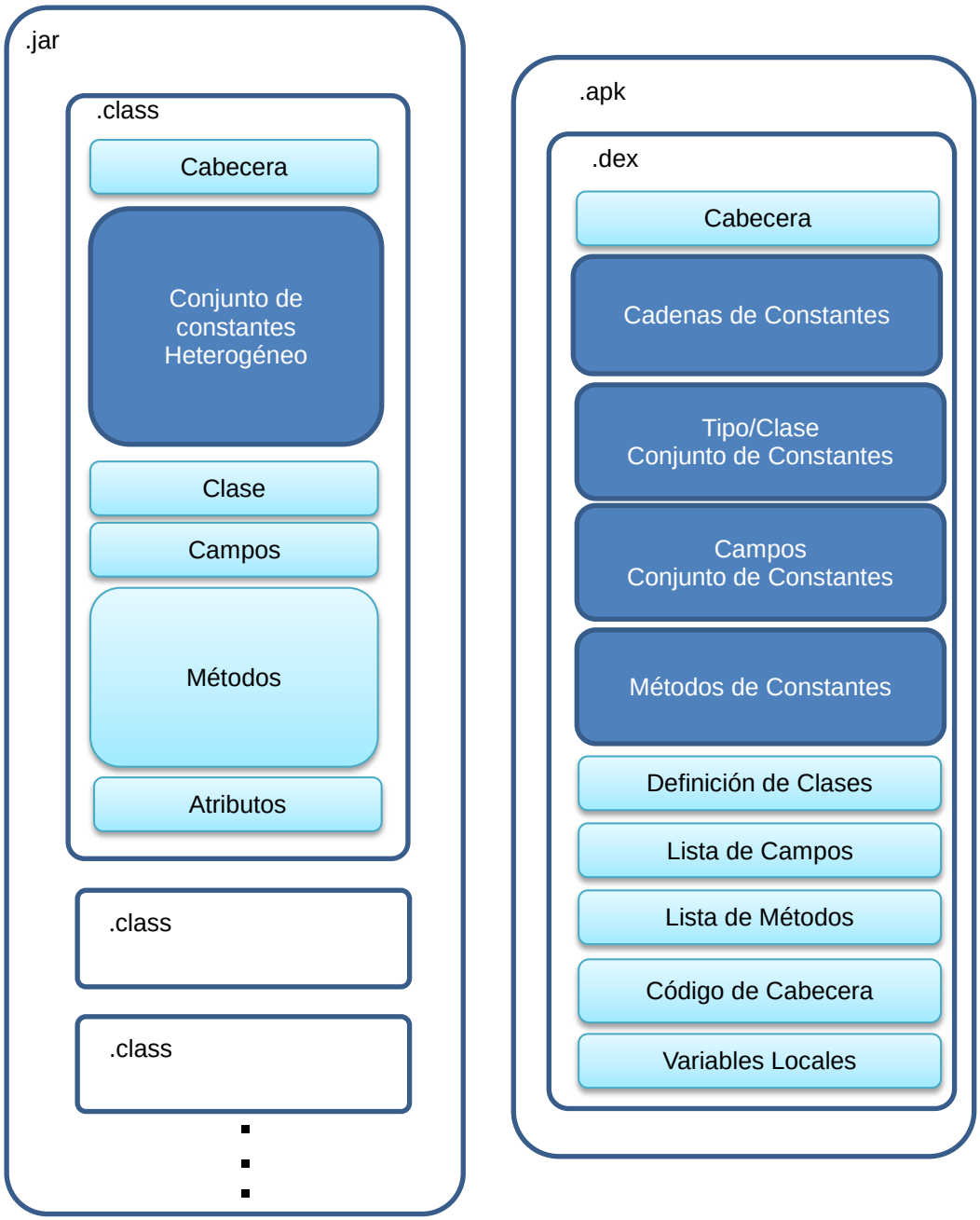

Figura 2.2.- Comparación entre archivos Class y Dex [13]

Dalvik permite reducir bastante el tamaño del programa buscando información duplicada en las diversas clases y reutilizándola. Esto le permite ahorrar una cantidad de memoria significativa. Según Google, el formato Dex corta en la mitad el tamaño de algunas de las librerías comunes del sistema y aplicaciones que vienen por defecto con Android. Podemos ver esta comparación en la tabla III, mostrando los tamaños en bytes de un mismo código, empaquetado con el formato Dex contra hacerlo en formato Jar, o un formato Jar comprimido [11], [13].

Lo que en Java llamamos “recolector de basura”, que libera el espacio en memoria de objetos que ya no van a ser utilizamos en nuestros programas, ha sido perfeccionado en Android con el propósito de mantener la mayor cantidad de memoria posible en todo momento. Es decir, Android posee un “recolector de basura” con dos tipos de colecciones [13]:

- Colecciones menores, que no usan mucho tiempo de procesador y son frecuentes, usados para recolectar objetos recientemente ubicados y objetos sin uso.
- Colecciones mayores, requieren de un mayor tiempo de procesador y son menos frecuentes, usados para recolectar

todos los objetos sin usos. Son realizados una vez que la memoria se agota.

Tabla III.- Comparación espacio de memoria entre Jar y Apk [13]

	Archivo JAR Descomprimido (bytes)	Archivo JAR Comprimido (bytes)	Archivo DEX descomprimido (bytes)
Librerías comunes del sistema	21,445,320 (100%)	10,662,048 (50%)	10,311,972 (48%)
Aplicación de un Navegador Web	470,312 (100%)	232,065 (49%)	209,248 (44%)
Aplicación de un reloj despertador	119,200 (100%)	61,658 (52%)	53,020 (44%)

2.4 Desarrollo de Aplicaciones en Android

Para desarrollar cualquier aplicación para Android, se debe conocer primero las herramientas a usar, en esta sección se explica cuáles son los kit de desarrollo que existen para Android el SDK y el NDK y para finalizar se explica que es el puente de depuración Android.

2.4.1 Kit de Desarrollo de Software Android (SDK)

Las aplicaciones son desarrolladas usando el lenguaje Java usando el Kit de Desarrollo de software Android (SDK), que incluye un

depurador, bibliotecas de software, un emulador de terminal basado en QEMU, documentación, código de ejemplo y tutoriales.

El IDE oficial soportado por Android es Eclipse acompañado por el complemento llamado Herramienta de Desarrollo Android (ADT). Otras herramientas de desarrollo disponibles, son el Kit de Desarrollo Nativo (NDK) para aplicaciones o extensiones en C o C++, Google App Inventor y varios marcos de trabajo inter-plataforma de aplicaciones móviles.

2.4.2 Kit de Desarrollo Nativo Android (NDK)

El NDK ayuda a implementar cierta parte de una aplicación usando lenguajes C y C++, usualmente referido como código nativo, porque es el mismo lenguaje con que está escrito el núcleo de android. El NDK brinda beneficios a ciertas aplicaciones, dado que se puede reutilizar el código fuente y en algunos casos obtener un aumento de la velocidad [21].

El NDK provee lo siguiente: Un conjunto de herramientas utilizadas para generar bibliotecas de código nativo de fuentes en lenguaje C y C++. Una manera de incluir dentro del paquete de la aplicación (apk) las correspondientes bibliotecas nativas. Las cabeceras y

librerías nativas que pueden ser soportadas por las futuras versiones de la plataforma Android, desde la versión 2.3 en adelante. Documentación, ejemplos y tutoriales [19].

Entre las librerías incluidas tenemos a la librería JNI (Java Native Interface) que sirve como un puente entre Java y Código Nativo, es decir, permite que las aplicaciones JAVA puedan incorporar código nativo escrito en lenguajes como C, C++ y Lenguaje Ensamblador, así como código escrito en el lenguaje de programación Java. Muchas de las librerías nativas de Android dependen de la funcionalidad que proporciona JNI para cumplir su propósito, por ejemplo, las librerías de Entrada/Salida, todo este funcionamiento se muestra en la figura 2.3.

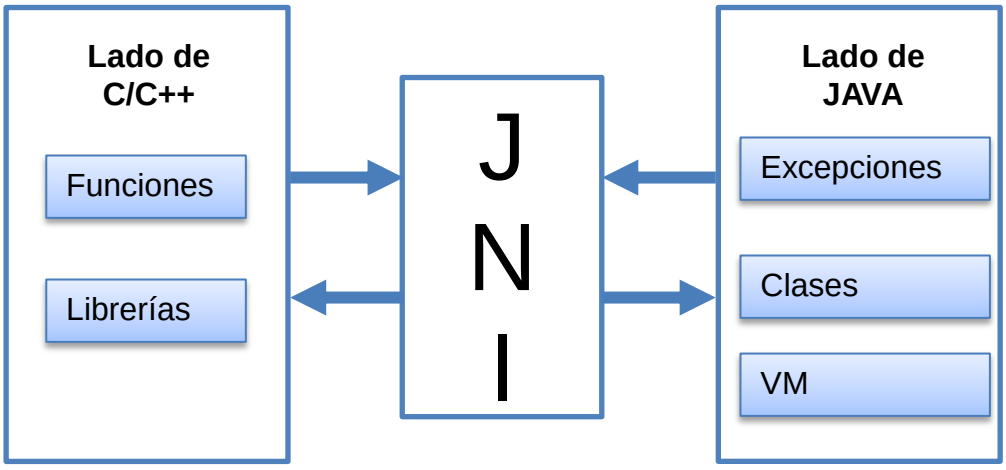

Figura 2.3.- JNI, el puente entre Java y Código Nativo

Otras librerías incluidas son libc, utilizada para las llamadas al sistema y otras funciones básicas como reservar memoria, presentar en pantalla, etc.; librería matemática, librería para compresión de información, librería de registros, librería de gráficos 2D y 3D.

2.4.3 Puente de depuración Android (ADB)

El ADB es un conjunto de herramientas incluido en el paquete de instalación del SDK. Consiste de programas cliente y servidor, cuyo acceso típicamente es a través de línea de comando, y que permite comunicar con un emulador o con un dispositivo Android [19].

Posee su propia sintaxis y comandos, permite acceder a la memoria y datos técnicos de nuestro dispositivo, instalar o eliminar aplicaciones, dependiendo de los permisos que se tenga sobre el dispositivo.

CAPITULO 3:
ARQUITECTURA DE ANDROID

Para poder desarrollar aplicaciones en Android, es importante conocer como está estructurado el sistema operativo, para el caso de Android su arquitectura se encuentra conformada de varias capas que facilitan al desarrollador la creación de aplicaciones. Esta distribución de Linux permite acceder a las capas de la arquitectura más bajas mediante el uso de librerías diseñadas para Android, esto para permitir que una aplicación haga uso del hardware de los teléfonos.

A la arquitectura de Android se la conoce como pila, esto es porque cada una de las capas puede acceder y utilizar elementos de la capa inferior. El diagrama de la arquitectura de Android se muestra en la figura 3.1 donde encontramos como la capa más baja al núcleo de Linux, luego las librerías, después los marcos de trabajo y por ultimo a las aplicaciones, en las

siguientes secciones explicamos cada una de las funcionalidades de estas capas empezando desde la capa inferior.

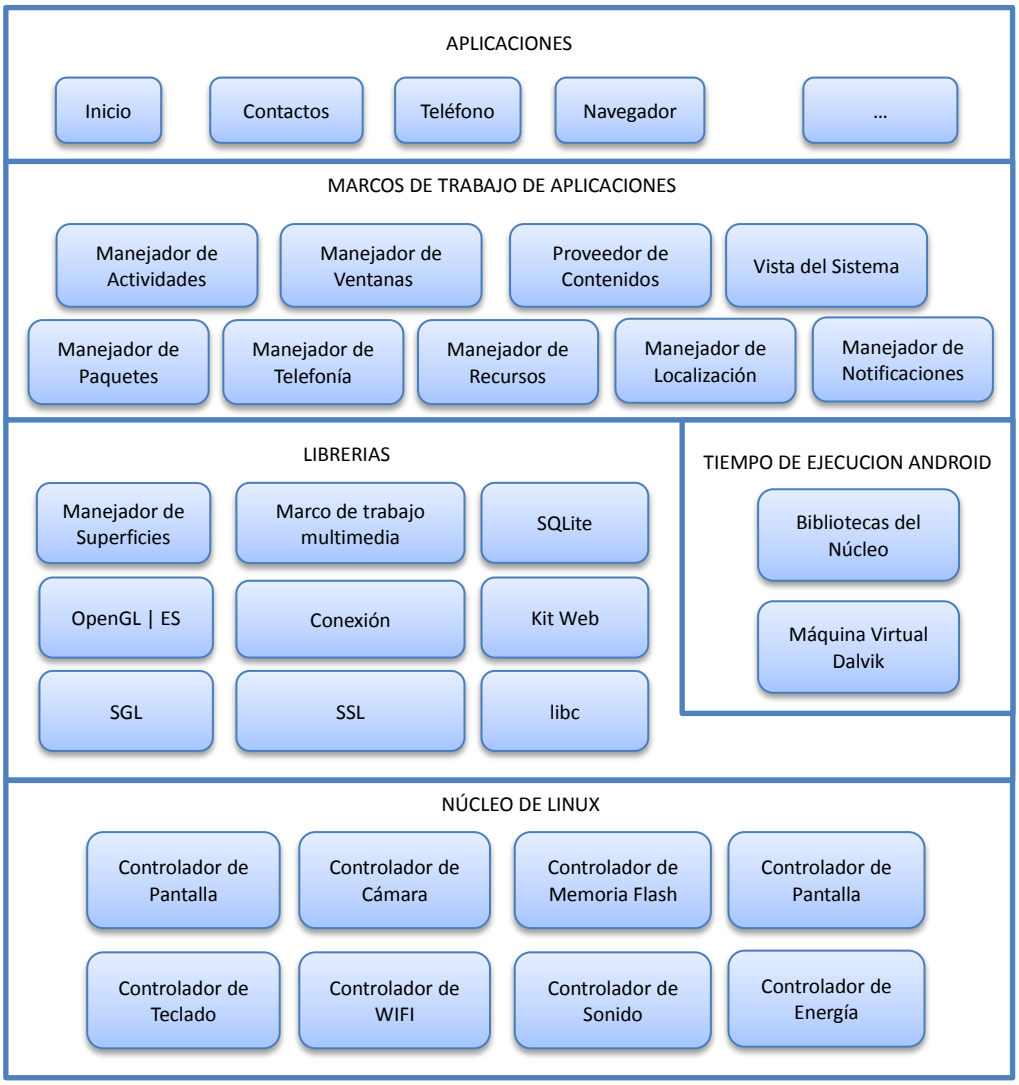

Figura 3.1.- Arquitectura de Android [15]

3.1 Núcleo de Linux

El Núcleo también se encarga de administrador los diferentes recursos del teléfono, como la energía, uso de memoria, etc., y de administrar el

sistema operativo en sí: procesos, elementos de comunicación (redes de trabajo), etc. Es una capa intermedia entre el hardware y el resto de la arquitectura. El desarrollador no debe preocuparse de acceder o desarrollar componentes para acceder al hardware del dispositivo, ni preocuparse de acceso a memoria, interrupciones o uso de procesador, esto lo hace a través de las librerías disponibles en capas superiores, de esta manera sin importar que hardware tenga cierto teléfono, siempre se lo podrá utilizar sin ningún inconveniente. Cada parte instalada en el teléfono posee un controlador (o driver) que permite manipularlo [8]. Por ejemplo, si es necesario utilizar la cámara integrada en el teléfono, el sistema operativo buscara y utilizará el controlador que viene incluido en el teléfono, sin importar sus características [16], [17].

Como dijimos al inicio de nuestro documento el núcleo del sistema operativo Android está basado en el núcleo de Linux versión 2.6, pero compilado para adaptarse a las características del hardware en el que se ejecutará el sistema operativo Android.

Android es una bifurcación del núcleo de Linux, por lo que evoluciona independientemente de los cambios o modificaciones que pueda tener el código original. Para modificar el núcleo de Linux se utilizaron varios parches (aproximadamente 250 parches, con cerca de 3 Megabytes y

25.000 líneas de código), el mayor es el sistema de archivo flash (YAFFS2) [18]. Podemos agrupar estos parches dependiendo de su principal objetivo, según el artículo de Linaro llamado “Android OS for Servers? [20]” tenemos los siguientes:

Los que fueron diseñados para el arreglo de errores y el poder habilitar nuevo hardware, como Pmem, que se encarga de asignar memoria, o el ADB (Puente de depuración Android) usado para un acceso más rápido a la funcionalidades del sistema; además de otros arreglos y modificaciones para las tarjetas de memoria, Bluetooth, etc. Los que se encargan del Manejo de Energía, suspensión temprana, temporizador de Alarma y mantenimiento de encendido. Los hechos para el manejo de Reporte de Errores, como el registrador que es utilizado para registrar los mensajes, la consola RAM, el Apanic que se encarga del registro de errores si un error del núcleo sucede. Los parches para la Seguridad, como el Red Paranoid por ejemplo, que restringe el acceso a algunas funciones de red en función del grupo del proceso que lo solicita. Los diseñados para mejorar el rendimiento, entre estos encontramos el Ashmen que maneja la memoria compartida anónima, para poder compartir datos entre procesos no relacionados; el Binder que es un mecanismo de comunicación entre procesos, y un sistema de invocación de método remoto, usado para que un proceso pueda mandar a ejecutar

una rutina en otro proceso; y el Manejo OOM que Permite matar procesos dependiendo de valores determinados por su ciclo de vida. Uno de los cambios más grandes fue el del sistema de archivo, se decidió utilizar el Yaffs2 fs, o Sistema de Archivo Flash, que esta libremente disponible para Linux, aunque no forma parte del núcleo estándar de Linux, el equipo de Google Android decidió agregarlo.

En marzo del 2012 LinusTorvalds y su equipo liberaron la versión 3.3 del núcleo de Linux, con la novedad que ya tiene integrado los parches y el código específico para Android. Esto implica que puede instalarse directamente una distribución de Linux sobre un teléfono que actualmente soporte Android y viceversa. Entre los beneficios que podemos mencionar está el hecho de que los fabricantes ahorraran tiempo y dinero al desarrollar controladores para sus dispositivos que pueden servir tanto en una pc con Linux, como en un teléfono con Android.

3.2 Librerías

La siguiente capa situada justo sobre el núcleo la componen las librerías nativas de Android. Estas son escritas en C o C++ y son compiladas específicamente para el hardware del teléfono. Normalmente son

desarrolladas por el fabricante y son instaladas en el teléfono antes de que sean distribuidos para la venta [16], [17].

Algunas bibliotecas que habitualmente se incluyen son la libc, el gestor de superficies, la librería de escenario gráfico, OpenGl/SL, la biblioteca multimedia, el Kit Web, la capa de conexión segura, el FreeType y el SQLite las cuales se describen a continuación [17]:

La Biblioteca C de sistema (libc) está basada en la implementación de BSD (Distribución de software Berkeley), pero es optimizada para su uso en sistemas Linux. Proporciona funcionalidad básica para la ejecución de las aplicaciones.

El Gestor de superficies es el encargado de la creación de las imágenes que se muestran en la pantalla. Cada vez que la aplicación pretende dibujar algo en la pantalla, esta librería no lo hace directamente sobre ella, en su lugar gestiona los cambios en imágenes que almacena en memoria para construir la imagen que finalmente se enviara a pantalla. La librería de Escenario Grafico (SGL) que es utilizada tanto en Android como en el navegador Google Chrome, esta librería es el motor gráfico 2D de Android, su función es la de presentar elementos en 2 dimensiones. Las librerías gráficas OpenGL/SL y SGL se encargan de

manejar gráficos en 3D y permite utilizar el hardware encargado de proporcionar gráficos 3D, en caso de que estén disponibles en el teléfono.

Para los gráficos en 2D, el encargado es SGL, y es el más utilizado en las aplicaciones. Para el audio y video, así como la visualización de imágenes, se utiliza la Bibliotecas multimedia permiten ver, reproducir y grabar una gran cantidad de formatos de imagen, audio y video (JPG, GIF, PNG, MPEG4, AVC, MP3, AAC o AMR).

Para la navegación se tiene el Kit Web que es motor base de navegadores como Chrome, Safari y Opera. Es un motor que se encarga de realizar el análisis sintáctico y de presentar en pantalla la información obtenida por una consulta web.

La seguridad de aplicaciones y datos del sistema operativo Android se realiza a través de SSL (Capa de Conexión Segura), que asegura el acceso a Internet por medio de criptografía. La librería FreeType permite mostrar fuentes tipográficas, basadas en mapas de bits o en representaciones vectoriales. Android posee un motor de base de datos integrado, el SQLite, disponible para todas las aplicaciones, muy sencilla de utilizar pero con poca seguridad.

3.3 Tiempo de Ejecución

El Tiempo de Ejecución de Android está basado en el concepto de máquina virtual utilizado en Java como lo explicamos antes, pero dado las limitaciones de los dispositivos sobre los que se ejecuta Android, no se pudo utilizar una máquina virtual Java estándar [14].

Como podemos apreciar en la figura 3.1, el Tiempo de Ejecución de Android provee un conjunto de librerías del núcleo, aquí se encuentran las librerías tanto Java como las específicas de Android. El principal componente de esta capa es la máquina virtual Dalvik, cuyo funcionamiento lo explicamos en el capítulo 2.4.

3.4 Marco de Trabajo de aplicaciones

Las aplicaciones utilizan clases y servicios que se encuentran en esta capa, la mayoría de los componentes disponibles en esta capa son librerías Java que acceden a los recursos disponibles en las capas inferiores a través de la máquina virtual Dalvik. Mencionamos a continuación los marcos de trabajo [16]:

El gestor de actividades se encarga de la administración, control del ciclo de vida y orden en el que se ejecutan las actividades, ventana de la

aplicación activa y cargada en memoria, a través de una pila de actividades.

El gestor de ventanas organiza y crea los espacios en pantalla que serán llenados por las actividades, haciendo uso del gestor de superficies. El proveedor de contenido encapsula y expone, de una manera ordenada y con la seguridad adecuada, los datos que se pueden compartir entre las aplicaciones. Los elementos que ayudan a crear las interfaces de usuario, como las cajas de texto, visor de imágenes, visor web o hasta mapa de Google, son llamados Vistas.

El gestor de notificaciones es una biblioteca que nos permite alertar al usuario de algún evento del sistema o de las aplicaciones ejecutándose, de una manera visual, en la barra de estado, manipulando los LEDS de teléfono (en caso de estar disponibles), haciendo vibrar el teléfono, o con una alerta sonora. El gestor de paquetes es el encargado de instalar, desinstalar y actualizar los paquetes de las aplicaciones, es decir, el apk de una aplicación.

El gestor de telefonía nos permite realizar llamadas y enviar/recibir mensajes de texto o multimedia. El gestor de recursos deja nos permite manipular los elementos que forman parte de la aplicación y que no

están definidos en código, como cadenas de texto traducidas a diferentes idiomas, imágenes, sonidos o diseños.

Si el teléfono posee un GPS incorporado, podemos hacer uso del mismo, y obtener información geográfica del mismo a través del Gestor de Localización. Otra manera de obtener esta información es mediante las redes disponibles, muy útil para trabajar con mapas.

La información de sensores disponibles en el teléfono puede ser obtenida a través del gestor de sensores, los más comunes son el giroscopio, acelerómetro, sensor de campo magnético, de luminosidad, de temperatura, de proximidad, de presión, la brújula, etc.

Dependiendo si el teléfono posee una o más cámaras, el uso de estas la podemos realizar a través de la librería diseñada para la manipulación de las cámaras. Los servicios Multimedia permiten reproducir y visualizar audio, vídeo e imágenes en el dispositivo.

3.5 Capa de Aplicaciones

La última capa incluye el conjunto de todas las aplicaciones disponibles del dispositivo, ya sean si poseen o no una interface de usuario, las nativas escritas en C o C++ y las que se encargan de administrar los

recursos del sistema, las que vienen instaladas por defecto las que el usuario instaló. En esta capa encontramos también la aplicación principal del sistema: Inicio (Home) o lanzador (launcher), que permite ordenar la ejecución de aplicaciones mediante un listado o como accesos directos en los diferentes escritorios, en estos también se puede colocar widgets [16], [17].

CAPÍTULO 4:
DISPOSITIVOS FUTUROS DE TECNOLOGÍA INTERNACIONAL Y DESARROLLO DE LA APLICACIÓN ANDROID

En esta sección se explica lo que es el módulo FTDI, también se muestra la forma de activarlo e instalarlo en el núcleo del teléfono, y se finaliza explicando el diseño y funcionamiento de la aplicación desarrollada.

4.1 Dispositivos Futuros de Tecnología Internacional (FTDI)

Los chips FTDI son chips desarrollados por la compañía escocesa de dispositivos semiconductores “Dispositivos Futuros de Tecnología Internacional”, que se especializan en la tecnología USB. Los adaptadores USB utilizan estos chips FTDI para la conexión de interfaces RS232 que es un estándar para el intercambio de datos binarios, y paralelo FIFO. USB-2-Interfaz COM es la interface más usada por los chips FTDI [23].

Sus circuitos integrados permiten que un equipo con puerto serie o paralelo pueda manejar dispositivos con interfaz USB tanto si tiene que actuar como esclavo, recibiendo instrucciones, o maestro, es decir controlando uno o más dispositivos. El chip FTDI lo utilizan sistemas basados en microcontroladores.

4.1.1 Integración con Android

Entre los circuitos integrados diseñados por la empresa FTDI, se tienen los chips FT311D y FT312D, un anfitrión USB que provee un puente automático desde tu puerto USB (Android) hacia un dispositivo externo a través de una interfaz de comunicación UART (Receptor Transmisor Universal Asíncrono).

La figura 4.1 muestra la comunicación entre el teléfono y un dispositivo externo por medio del módulo FTDI, el puerto usb del teléfono se conecta al módulo FTDI por medio de los pines USB D+ y el USB D- que son los encargados de enviar y recibir la información, luego el módulo FTDI se conecta al dispositivo externo por medio del módulo UART que es el encargado de recibir la información en formato paralelo y transformarla a formato serie y viceversa, haciendo uso de los siguientes pines, el TXD que recibe los datos, RXD que envía los datos, el RTS y el CTS que controlan

el flujo de información, el RTS que solicita el envío de información y el CTS que indica que el UART está disponible para enviar información.

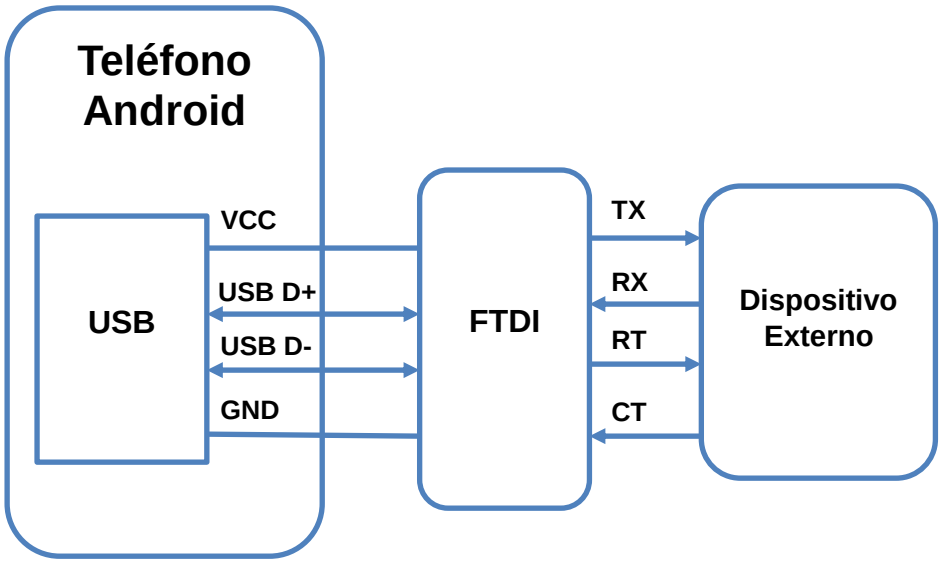

Figura 4.1.- Comunicación del teléfono mediante el FTDI

Como se aprecia en la figura 4.2, el chip FT312D posee un módulo UART que provee la transmisión y recepción básica, como se aprecia en los pines TXD y RXD, adicionalmente se puede ver los pines CTS y RTS, que se usan para controlar el flujo de información de tal manera que no haya perdida de datos. Se usa un buffer para evitar que se pierda información mientras se establece comunicación entre el USB Anfitrión y el módulo UART. El bloque del USB anfitrión se encarga de manejar la conversión serial a paralelo y paralelo a serial de la capa física USB, incluyendo la

generación de crc (comprobación de redundancia cíclica). Finalmente el transceptor USB provee la interfaz física del dispositivo USB, soportando estándares USB 1.1 y USB 2.0, lo que se representa con los hilos USBDP (D+) y USBDM (D-), que son los pines por los cuales se realiza la transmisión USB.

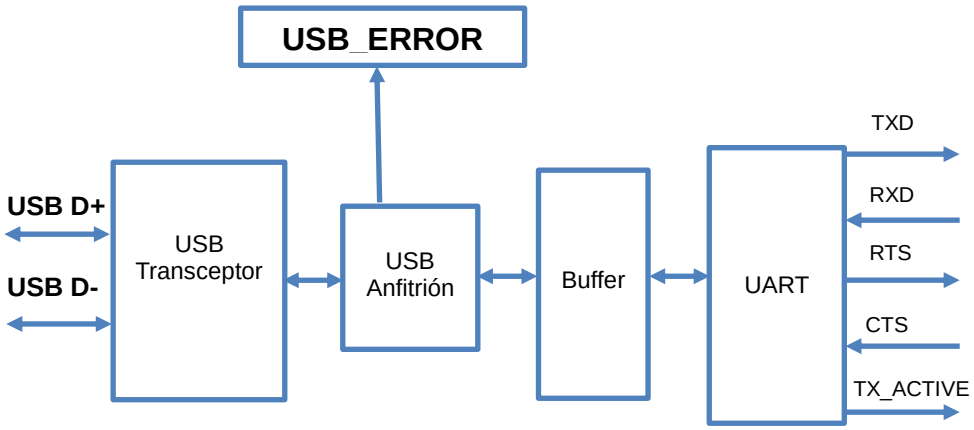

Figura 4.2.- Diagrama de Bloques del chip FT312D

4.1.2 Soporte USB OTG (USB Anfitrión)

Muchos teléfonos con Android ahora tienen puertos USB OTG (On the go), permitiendo a los puertos USB ser un anfitrión que provee energía al dispositivo conectado, o un invitado, como al conectarlo a una computadora para que este funcione como un dispositivo de almacenamiento externo.

Algunos de los dispositivos de última generación tienen el OTG completamente habilitado, pero el principal problema con los

puertos anfitriones es que un periférico en particular requiere un controlador especial para que sea cargado, lo cual significa que el propietario del dispositivo debe dar los permisos de superusuario (root) para obtener los permisos necesarios para permitir la instalación de nuevos controladores.

OTG es una extensión del USB 2.0 que permite conectar dispositivos con el fin de usar al móvil como anfitrión. Con esta extensión se pueden conectar teclados, discos externos, mandos, discos duros y otros dispositivos, y esto se consigue conectando un pequeño adaptador al móvil Android. Casi todos los dispositivos externos funcionan recibiendo 5V, que es lo que suele dar un puerto USB corriente, pero si no disponemos de USB OTG de fábrica, nuestro puerto no nos dará el voltaje necesario para hacer funcionar el dispositivo externos [26].

4.2 Habilitar módulo FTDI en Android

En esta sección se explican los pasos que se debe seguir para habilitar el módulo FTDI en el teléfono, se comenzó por obtener la correcta versión del código fuente del núcleo del teléfono, luego se obtuvo permisos de superusuario, después se configuró y compiló el núcleo de android, para finalmente instalar el módulo FTDI en el teléfono. La figura

4.3 muestra los pasos que se siguió para poder habilitar el módulo FTDI en el sistema operativo android.

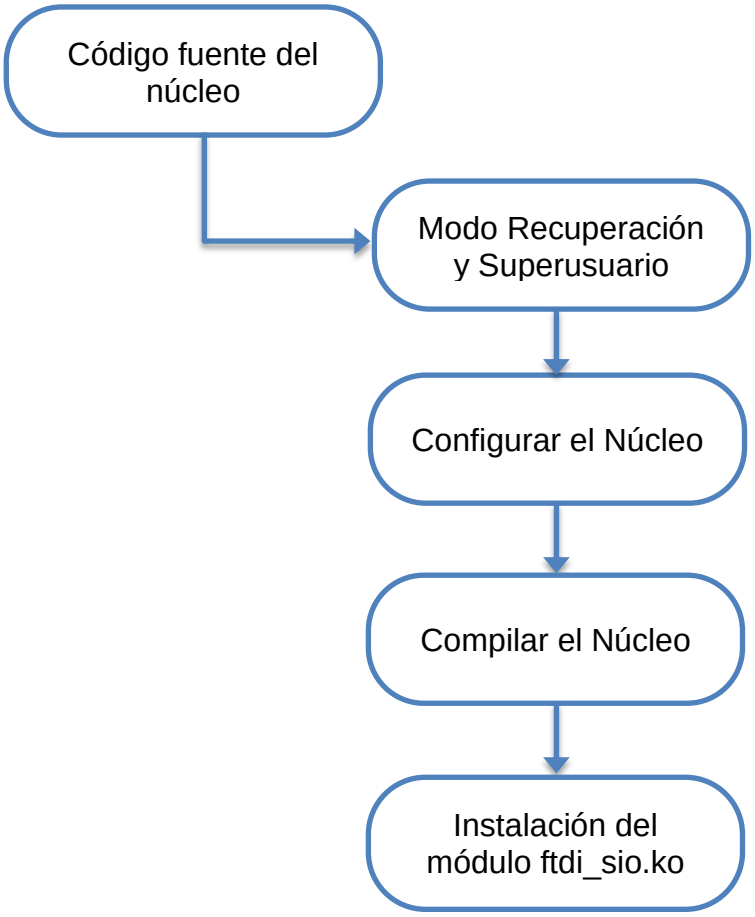

Figura 4.3.- Pasos para habilitar módulo FTDI en android

4.2.1 Código fuente del Núcleo

El Código fuente del núcleo es un grupo de archivos y carpetas, en este grupo de carpetas se encuentra en el directorio “Drivers”, en esta carpeta están los módulos que se pueden instalar al núcleo, entre ellos el modulo que se necesitó para activar el FTDI en el

núcleo, este es el archivo ftdi_sio.ko.

En nuestro caso se utilizó el núcleo para la tableta Samsung Galaxy Tab 2 GT-P3113, que se puede encontrar en el sitio web de “Centro de liberación de código libre de Samsung”, http://opensource.samsung.com/.

4.2.2 Modo Recuperación y Superusuario

El superusuario, también llamado root, es el usuario que comúnmente se usa para tareas administrativas, como configurar el sistema operativo o instalar un software, y puede acceder a cualquier parte del sistema operativo. En Android la ROM es un archivo en formato zip que guarda la configuración del sistema o el programa de arranque del teléfono.

Para realizar modificaciones en el núcleo, instalar roms o habilitar módulos en el núcleo se necesita tener privilegios de superusuario, debido a que los usuarios de los teléfonos pueden alterar el núcleo de manera errónea los fabricantes de celulares mantienen bloqueado el acceso como superusuario a sus clientes, y se necesita instalar archivos adicionales en el teléfono para poder acceder como este usuario.

Para habilitar el superusuario en un teléfono inteligente con sistema operativo Android se debe ingresar al modo recuperación cuya interfaz se muestra en la figura 4.4, es un modo donde se puede dar mantenimiento al teléfono es decir limpiar la cache, realizar copias de seguridad del sistema y permite restaurar estas copias del sistema [27].

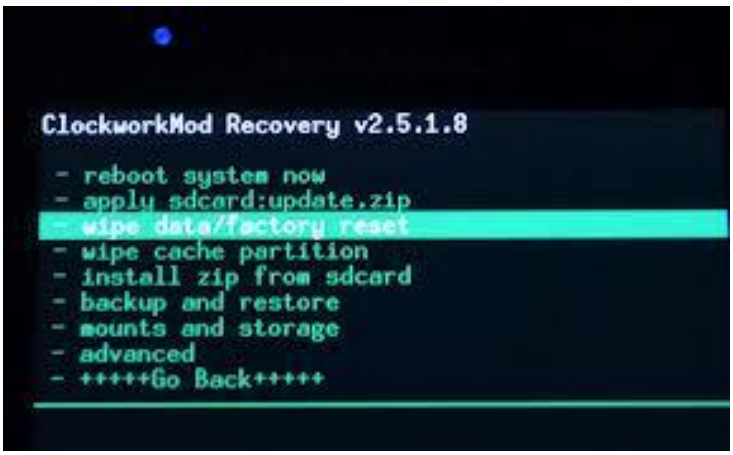

Figura 4.4.- Pantalla del Modo Recuperación

Para poder acceder como superusuario en la Tablet Samsung Galaxy Tab 2 GT-P3113 se modificó el núcleo con el software “Odín” que nos permite cambiar toda o una parte del núcleo del teléfono con la finalidad de mejorar las funciones del núcleo [17], después se inició el teléfono en modo recuperación manteniendo presionado las teclas INICIO + VOLUMEN ALTO, y se instaló el archivo llamado “cwm-root-gtab2.zip”. Posteriormente se ingresó al teléfono y se verificó que ya se encontraba instalada la aplicación “Superusuario”

que se muestra en la figura 4.5, esta se encarga de otorgar los permisos de superusuario a cada una de las aplicaciones que lo soliciten.

Al modificar el núcleo del teléfono con Odín debemos tener cuidado, ya que existe el riesgo de hacer un cambio incorrecto en el núcleo y dejar inutilizable el teléfono, ya que en nuestro caso se desconectó el teléfono cuando se encontraba en el proceso de modificación del kernel y dicho teléfono no volvió a encender.

Figura 4.5.- Aplicación Superusuario

4.2.3 Configuración del Núcleo

La configuración del núcleo se realizó en una computadora personal y lo primero que se tuvo en cuenta es tener el archivo correcto de configuración ".config" para nuestra versión de núcleo, una forma de

obtener este archivo es extrayéndolo del teléfono con el comando:

```
$ adb pull /proc/config.gz config.gz
```

Este comando obtiene una copia del archivo de configuración “config.gz” del teléfono, este se encuentra en la ruta /proc/config.gz, y lo guarda en el computador. Para reemplazar el archivo de configuración “config.gz” por el archivo “.config” del núcleo se usa el siguiente comando:

```
$ zcat config.gz > ~/kernel/.config
```

Algunos teléfonos, en la configuración de su núcleo no tienen habilitado la opción de mostrar el archivo “config.gz”, por lo que no se puede extraer la configuración de estos teléfonos. Para este caso se debe cargar un archivo del núcleo que se configuró, estos archivos de configuración están presentes en la ruta ~/kernel/arch/arm/configs/. En nuestro caso para la tableta Samsung Galaxy Tab 2 GT-P3113 el archivo de configuración es “android_espresso_omap4430_r04_user_defconfig” y se habilitó con el siguiente comando:

```
make ARCH=arm
android_espresso_omap4430_r04_user_defconfig
```

El Menú de configuración del núcleo mostrado en la figura 4.6, es una interfaz de usuario, se presenta como un menú con todas las opciones para configurar el núcleo, este es el paso previo a la compilación, en esta interfaz se habilitó los módulos del núcleo y nuestro objetivo fue el módulo FTDI.

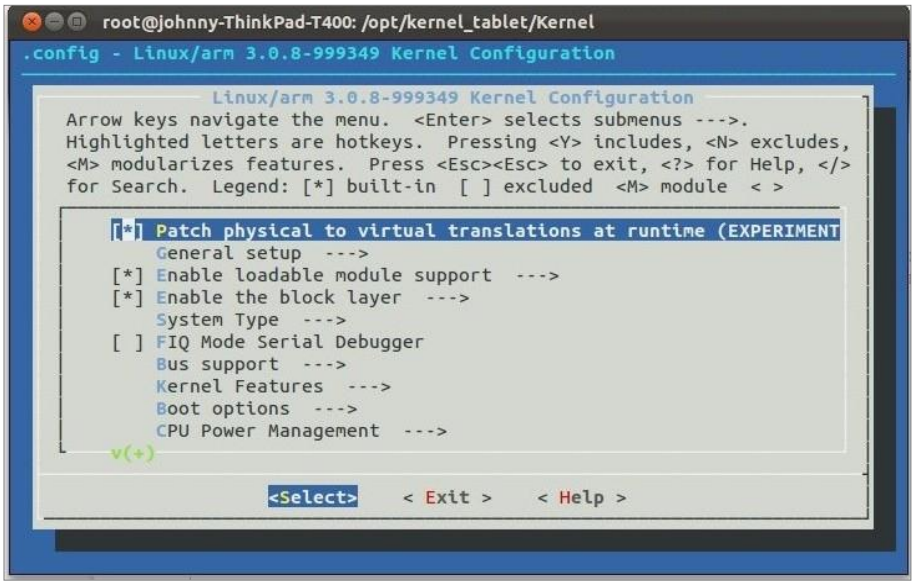

Figura 4.6.- Menú de configuración

Para acceder al menú de configuración se ejecutó el siguiente comando:

```
make menuconfig ARCH=arm
```

Para habilitar este módulo se navegó por los submenús del menú de configuración, y se ingresó a la siguiente jerarquía:

“Controladores de Dispositivos -> Soporte USB -> Soporte

Convertidor Serial -> Controlador Simple de Puerto Serial FTDI”.

En este menú se activó el módulo FTDI colocando a la izquierda de “Controlador Simple de Puerto Serial FTDI” la opción “<M>” como lo muestra la figura 4.7.

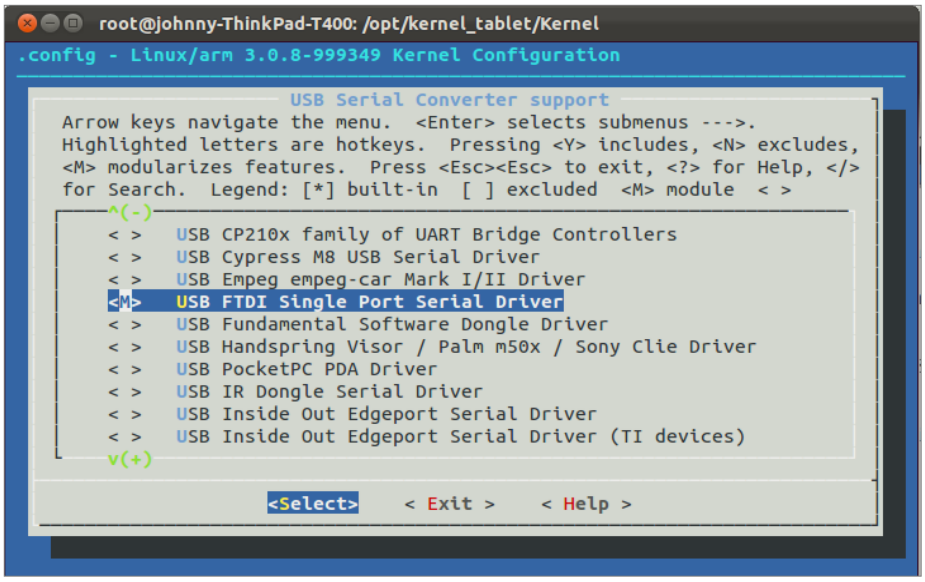

Figura 4.7.- Controlador Simple de Puerto Serial FTDI

4.2.4 Compilación del Núcleo

En ocasiones los núcleos de fábrica que tienen los teléfonos contiene una “versión extra”, y se debe ajustar el archivo Makefile con la versión correcta del núcleo, en nuestro caso se tenía la versión “3.08-999349” y se editó el archivo makefile, colocando la línea “EXTRAVERSION= -999349”.

Para compilar el núcleo de Android se necesitó un compilador de lenguaje “C”, en nuestro caso se usó el que viene integrado en el NDK, se exportó la ruta de donde se encontraba el compilador de la siguiente forma:

```
export PATH=/opt/android/android-
ndk/toolchains/arm-linux-androideabi-
4.4.3/prebuilt/linux-x86/bin/:$PATH
```

Se realizó la compilación con el fin de generar todos los módulos del núcleo de Android que se habilitaron en la configuración del menú, también para generar la imagen del núcleo que se puede instalar en nuestro teléfono, por medio del software “Odín”.

Para compilar el núcleo se ejecutó el siguiente comando:

```
make ARCH=arm CROSS_COMPILE=arm-linux-androideabi-
-j4
```

La opción CROSS_COMPILE indica el compilador que se utilizó, en nuestro caso el NDK. Al compilar el modulo en el teléfono se presentó el siguiente error “ftdi_sio.ko: Unknown symbol _GLOBAL_OFFSET_TABLE” debido a esto se necesitó compilar los módulos con la bandera “EXTRA_CFLAGS=-fno-pic”, todos los

módulos del núcleo no aceptan esta bandera por lo que se presentaron errores al compilar, por esto solo se compiló los módulos que se necesitaron y en los que se dieron estos errores, por esto se especificó la ruta de los módulos a compilar, el comando se ejecutó de la siguiente forma:

```
make ARCH=arm CROSS_COMPILE=arm-linux-androideabi-
-j4 EXTRA_CFLAGS=-fno-pic M=drivers/usb/serial
```

En la carpeta “serial” se encontró el archivo “ftdi_sio.ko” que se necesitó para activar el módulo FTDI del núcleo de android. Determinados teléfonos no cuentan con los módulos USBCORE y USBSERIAL que son necesarios para que se pueda activar el módulo FTDI, por lo tanto también estos módulos deben ser compilados para que se generen los archivos “usbcore.ko” y “usbserial.ko”. Cabe resaltar que en nuestro caso no se necesitó compilar ni instalar estos dos módulos adicionales.

4.2.5 Instalación del módulo FTDI en Android

Se transfieren los archivos al teléfono en algún directorio conocido por medio del módulo ADB, en nuestro caso se los transfirió al directorio “/sdcard” del teléfono, para esto se conectó el teléfono al computador y se ejecutó:

```
adb push drivers/usb/serial/ftdi_sio.ko /sdcard/
```

Se utilizó el FTDI para comunicación mediante radios, para esto se instaló el archivo “ftdi_sio.ko” en el sistema android del teléfono inteligente. Una vez conectado el teléfono al computador, se ingresó al terminal del teléfono con la ayuda del ADB con el siguiente comando:

```
adb Shell
```

Para la instalación del módulo se localizó el directorio en donde se guardaron los módulos, en nuestro caso el archivo “ftdi_sio.ko” que se ubicó en el directorio “/sdcard”, se cambió a modo superusuario con el comando “su” y ejecutamos:

```
insmod ftdi_sio.ko
```

Con este comando el módulo se cargó en el teléfono, para asegurarse que se habilitó el módulo, se ejecutó “lsmod” y se mostró una lista con los módulos habilitados en el sistema, y en nuestro caso apareció en la lista la línea “ftdi_sio 30137 0- Live 0xbf072000”, esto indicó que el modulo se había instalado correctamente como lo muestra la figura 4.8.

En el caso de que el teléfono no tenga instalado los módulos USBCORE y USBSERIAL, se debe proceder a instalar primero estos módulos, en el siguiente orden primero el archivo “usbcore.ko”, luego el “usbserial.ko” y al final el “ftdi_sio.ko” todos ellos con el comando “insmod”.

Figura 4.8.- Instalar y listar módulos del núcleo

4.2.6 Errores al instalar Módulos

Al instalar el módulo se dio un error de versión del núcleo debido a que el núcleo que se descargó no era el correcto para el teléfono, y en otra prueba que se realizó el núcleo no era para la arquitectura de hardware del CPU.

Estos errores se pudieron observar al ejecutar el comando “cat” en el archivo donde se guardan los registros del sistema es decir

“/proc/kmsg”, el comando se ejecutó de la siguiente forma:

```
cat /proc/kmsg
```

Una vez que se dio el error se observó en el archivo “kmsg” el error:

```
ftdi_sio: version magic '2.6.32.27 preempt
mod_unload ARMv5' should be '2.6.32.27 preempt
mod_unload ARMv7’.
```

En este error la versión del núcleo era la correcta pero no la arquitectura, el núcleo descargado era para una arquitectura ARMv5 y el teléfono tenía una arquitectura ARMv7.

Una vez realizado todo el proceso de instalación del archivo “ftdi_sio.ko” el teléfono se habilitó para usar el módulo FTDI, y al conectar la radio al teléfono, en el directorio “/dev” que contiene los archivos de dispositivos que permiten la comunicación con el hardware que tengamos en el teléfono, apareció el archivo “ttyUSB0” que indicó que la radio estaba conectada al teléfono y podía empezar a transmitir o recibir datos.

4.3 Diseño de la Aplicación

La aplicación accede al módulo FTDI del teléfono y obtiene tramas de

manera constante, estas tramas se envían a través de un socket desde un proceso “C” a un proceso “Java” donde se verifica la validez de la trama y en caso de ser válida se grafica en un mapa de google. Se desarrolló un primer diseño de la aplicación el cual tuvo inconvenientes principalmente en la interfaz de usuario por lo que fue necesario crear un segundo diseño que fue el definitivo. A continuación explicamos los dos diseños desarrollados.

4.3.1 Primer Diseño

El primer diseño constó de dos procesos: uno en “java” y uno en “C”, que se comunicaban mediante el JNI, el proceso en “java” llamaba a una función nativa escrita en lenguaje “C” que abría un puerto mediante el módulo FTDI hasta obtener un dato, mientras el proceso “Java” esperaba la respuesta de esta función, una vez que había respuesta el proceso en “Java” continuaba con su ejecución.

El problema que se presentaba en esta versión es que al llamar a la función en lenguaje “C” el proceso en “Java” permanecía bloqueado hasta recibir una respuesta, y al estar la interfaz de usuario en este proceso “Java” esta quedaba bloqueada.

Este diseño no llegó a ser la solución debido al bloqueo de la

interfaz de usuario. La primera versión del diseño se muestra en la figura 4.9.

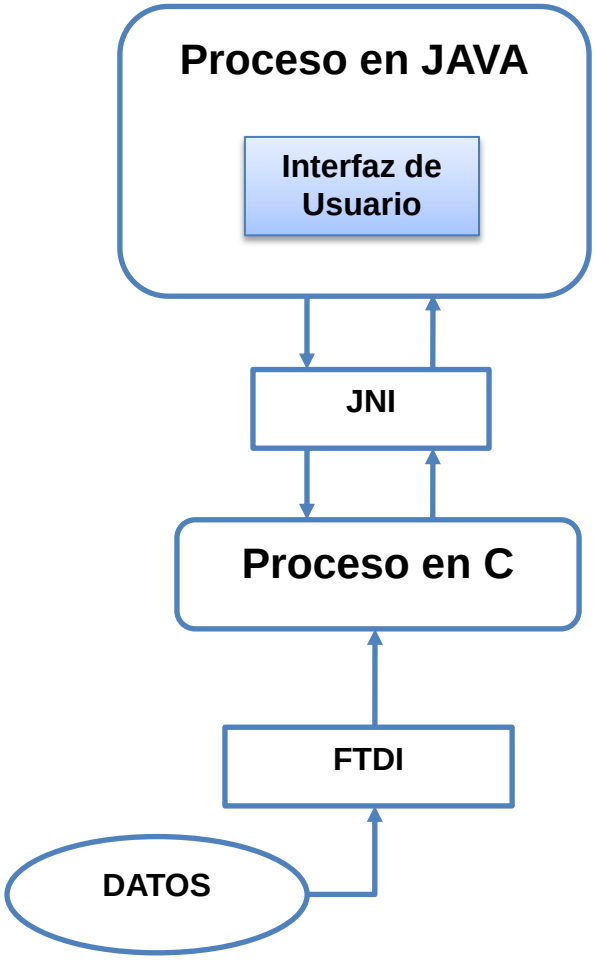

Figura 4.9.- Primer Diseño de la Aplicación

4.3.2 Segundo Diseño

Esta fue la versión final del diseño que constó al igual que el primer diseño de 2 procesos uno en “Java” y otro en “C”, el proceso en “Java” contenía la interfaz del mapa y cuyo código fuente se encontraba en el archivo “Mapa.java”, el proceso en “Java” llamaba

a un método nativo en lenguaje “C” llamado “leer” que se encontraba implementado en la librería nativa “milibreria” y que contenía el archivo “native.c”, en este archivo se encontraba el código del proceso en “C”.

El método “leer” creaba un hilo, que es un proceso de ejecución concurrente que consta de su propia pila, argumentos y variables locales, de nombre “iniciar_socket” en donde se realizó todo el proceso de lectura de datos mediante el módulo ftdi, luego el método finalizaba sin retornar valor alguno al proceso en “Java”, es decir retornaba a la interfaz donde se podía manipular el mapa sin bloquear la aplicación.

El hilo permanecía en ejecución y accedía al dispositivo a través del archivo “ttyUSB0” para poder realizar la lectura de los datos que llegaban al puerto.

```
puerto = open("/dev/ttyUSB0", O_RDWR | O_NOCTTY |
O_NONBLOCK);
```

Para comunicar los procesos entre “C” y “Java” se utilizó un socket servidor en el proceso “C” el cual en nuestro caso utilizaba el puerto 5555, en este socket se enviaban los datos leídos del archivo

“ttyUSB0”, y para recibir los datos en el proceso “Java” se utilizó un socket cliente pero que se conectaba al mismo teléfono es decir al “anfitrión local” (localhost).

El hilo en el proceso “C” enviaba los datos de las coordenadas de manera constante, por este motivo el proceso “Java” debía tener una tarea que tome esos datos de la misma manera, para esto se utilizó un proceso asíncrono mediante la creación de la clase “MiTareaAsincrona” que extiende de la clase “AsyncTask”.

El proceso asíncrono “MiTareaAsincrona”, contenía al socket que recibía los datos, luego los depuraba y enviaba las coordenadas a la interfaz de usuario que procedía a mostrarlas en el mapa.

El problema que se presentaba en la primera versión ya no se daba en la segunda versión ya que al crear un hilo la ejecución continuaba sin bloquear el proceso en “Java”, y los datos se transmitían por medio del socket de comunicación, siendo esta versión la solución final del diseño. El segundo diseño de la aplicación se muestra en la figura 4.10.

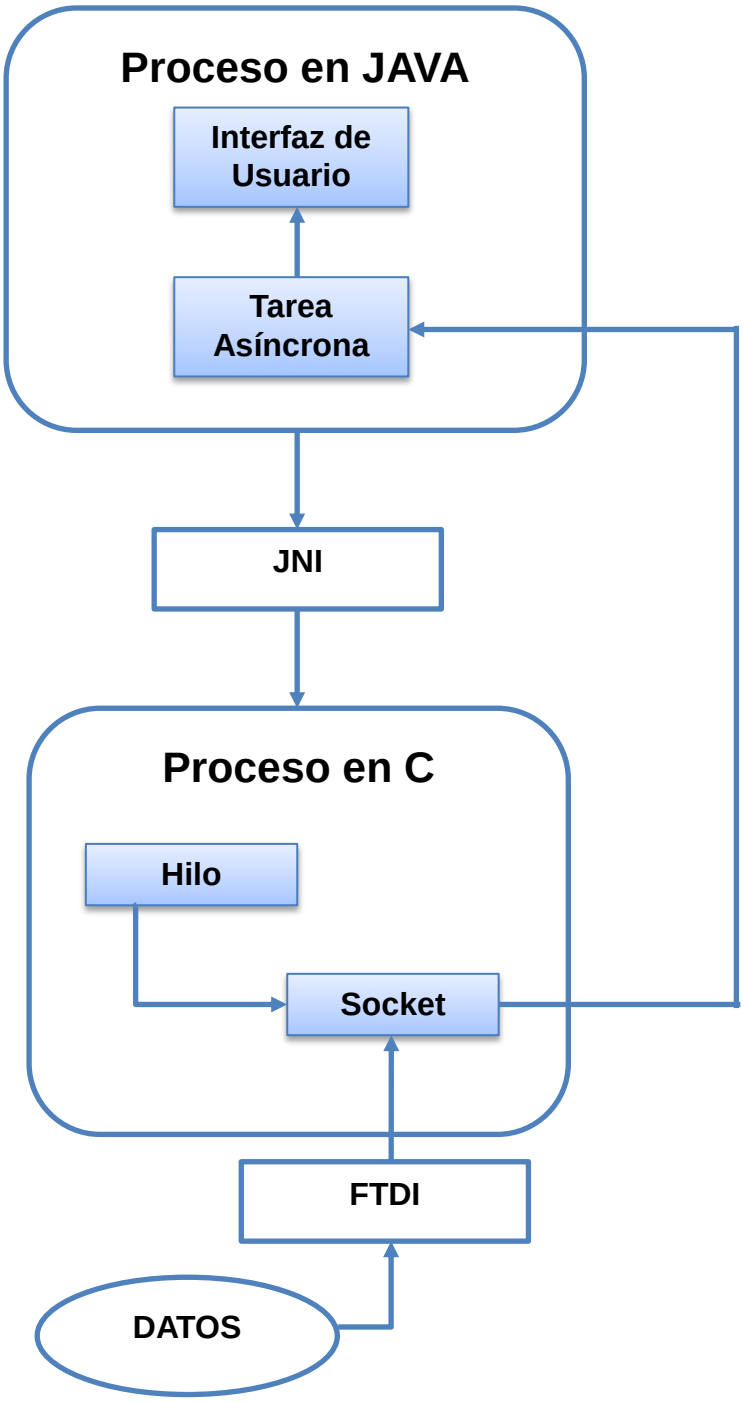

Figura 4.10.- Segundo Diseño de la Aplicación

4.4 Implementación

En esta sección se muestra como dar permisos de lectura y escritura al dispositivo que se conecta mediante el FTDI al teléfono, luego se muestra la interfaz que se utiliza para la comunicación entre los procesos “JAVA” y “C”, y la creación de la librería que contiene las funciones para dicha comunicación y al final se explica el funcionamiento de la aplicación.

4.4.1 Permisos para el Dispositivo

Para acceder al dispositivo a través del archivo “ttyUSB0” desde la aplicación android se necesitó tener permisos de lectura y escritura. Para poder dar estos permisos desde la aplicación, primero se obtuvo permisos de superusuario, para esto se utilizó la librería “RootTools.jar” y se ejecutó el siguiente comando:

```
RootTools.isRootAvailable()
```

Al tener permisos de superusuario en la aplicación se pudo darle permisos al dispositivo “ttyUSB0” para esto se usó la misma librería “RootTools.jar” y se ejecutó el siguiente código:

```
CommandCapture command = new CommandCapture(0,
"chmod 666 /dev/ttyUSB0");
try{
   RootTools.getShell(true).add(command).waitForFi
```

```
nish();
}catch (Exception e) {
   e.printStackTrace();
}
```

Al ejecutar el comando "chmod 666 /dev/ttyUSB0” se dio permisos 666 que significa que tanto los usuarios propietarios, grupos y otros obtuvieron acceso de lectura y escritura al archivo. Si no se obtiene estos permisos no se podrá leer del archivo “ttyUSB0” y por ende la comunicación no se realizara.

4.4.2 Interfaz de comunicación entre Java y C

Para tener una comunicación entre los procesos de “Java” y “C”, se declaró en el archivo “Java” el prototipo de un método nativo de “C”, en nuestro caso se declaró el siguiente método:

```
private native void leer();
```

La librería que contiene el método anterior se cargó en una inicialización estática, en nuestro caso la librería “milibreria”.

```
Static {
   System.loadLibrary("milibreria");
}
```

Se creó un directorio con el nombre “jni” donde se guardó el archivo “C” que contiene el método nativo, también en este directorio se

guardó el archivo “Android.mk” que contiene la información de compilación de la librería nativa “milibreria” y el archivo “native.c”.

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_LDLIBS := -llog
LOCAL_MODULE    := milibreria
LOCAL_SRC_FILES := native.c
include $(BUILD_SHARED_LIBRARY)
```

En el archivo “native.c” es donde se implementó el método nativo “leer”, y la función está compuesta por el tipo de dato que devuelve, el nombre del paquete de la aplicación en nuestro caso “com_espol_mapa”, el nombre de la actividad que es la clase donde se encuentra el código del proceso “Java” en nuestro caso “Mapa” y el nombre del método.

```
void Java_com_espol_mapa_Mapa_leer(JNIEnv * env,
jobject this)
{
}
```

Para compilar la librería se ingresó al directorio raíz del proyecto en nuestro caso “cd ~/workspace/Mapa”, y se ejecutó el comando de compilación “ndk-build”.

El ndk-build generó un archivo compilado de la librería, en la ruta “Mapa/libs/armeabi/libmilibreria.so” como se muestra en la figura 4.11, al generar el empaquetado de la aplicación, “Mapa.apk” esta

librería también se añadió a este empaquetado y se instaló junto con la aplicación en el teléfono.

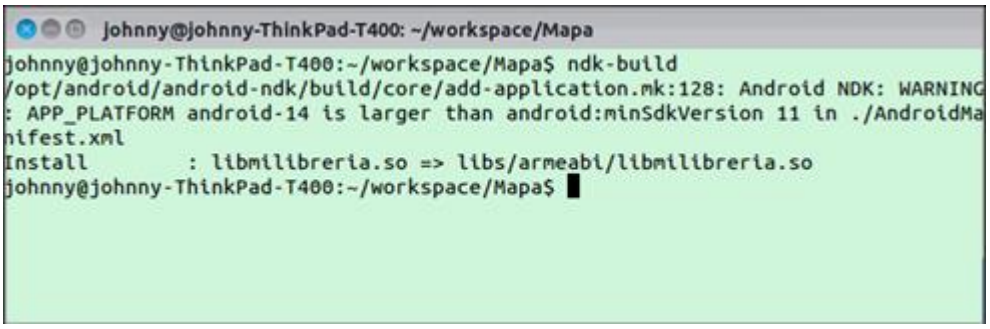

Figura 4.11.- Compilación de la librería nativa

4.4.3 Funcionamiento de la Aplicación

Al iniciar la aplicación además de presentar la pantalla principal, se obtienen los accesos de superusuario y se configura los permisos de lectura y escritura para el archivo “ttyUSB0”. Después que se obtiene los permisos se accede a la pantalla del mapa al presionar el botón comenzar.

Una vez dentro se puede ver el mapa de google, un botón para cambiar el tipo de mapa que se presenta y otro botón para iniciar y detener la toma de datos.

Al momento de presionar el botón iniciar se espera a que las tramas con las coordenadas geográficas empiecen a llegar. La trama está

compuesta de los valores de coordenadas, la información del punto geográfico y el checksum al final de la trama, el cheksum es un código hexadecimal que se calcula mediante una operación exor entre cada uno de los caracteres.

Se procede a calcular el checksum de la trama recibida y se lo compara con el checksum recibido. Si los valores son diferentes se considera una trama inválida y se descarta, quedando a la espera de la próxima trama.

Si los checksum coinciden la trama es válida y se procede a transformar las coordenadas recibidas que están en formato NMEA a formato Decimal, luego se dibuja el punto en el mapa y se queda a la espera de la próxima trama.

En todo momento de la ejecución si se presiona el botón detener cualquier trama recibida no será dibujada y finalizara el proceso de obtención de datos. La figura 4.12 muestra el funcionamiento de la aplicación desde la captura de las tramas hasta la presentación del punto en el mapa.

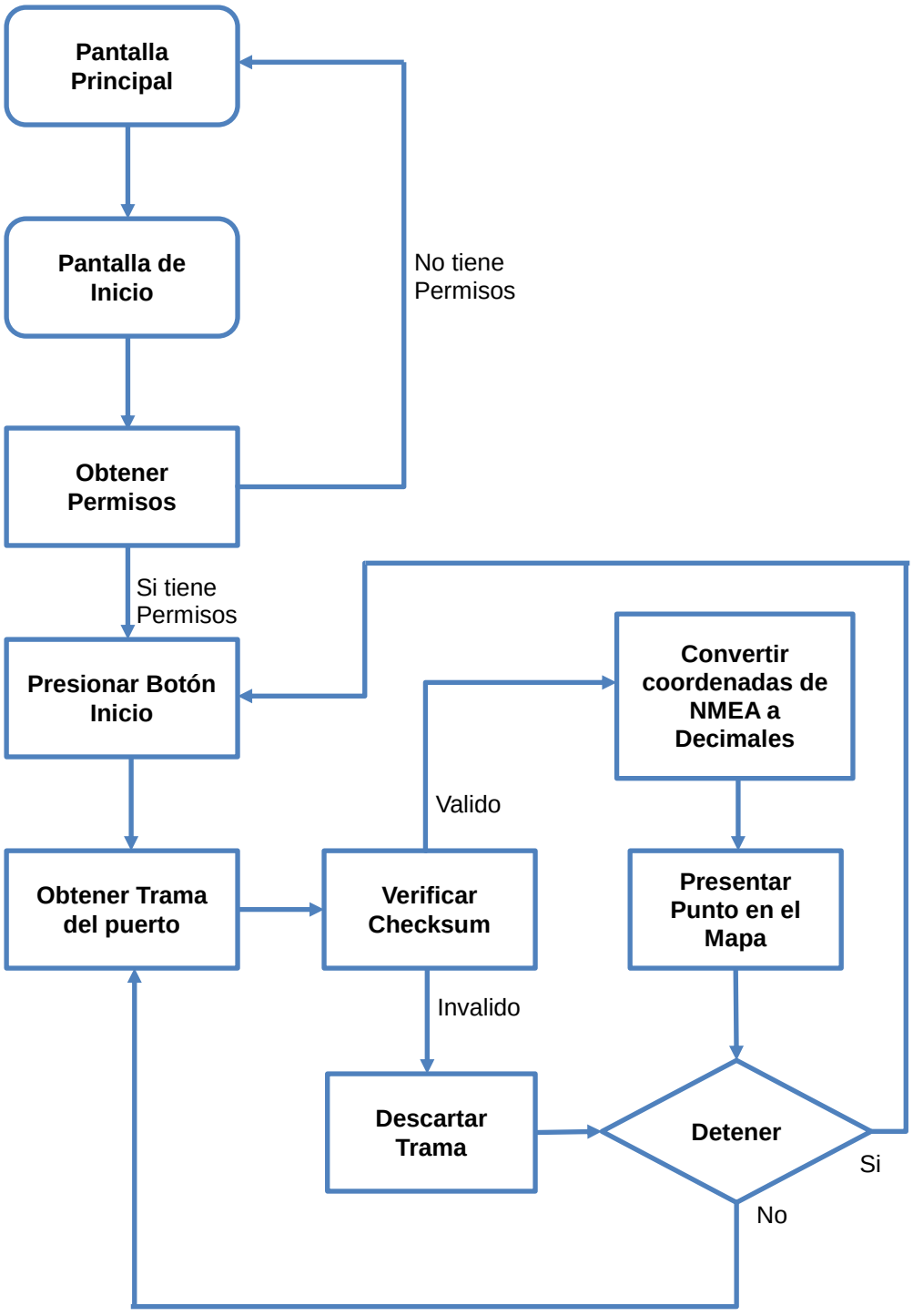

Figura 4.12.- Funcionamiento de la Aplicación

4.5 Experimento

Para los experimentos se conectó una de las radios Xbee a nuestra computadora portátil, esta funcionó como servidor y se encargó de transmitir las tramas con las coordenadas geográficas. La otra radio se conectó al teléfono y esta se encargó de recibir las coordenadas. Para medir la efectividad de las radios se modificó la frecuencia del envió de datos y la distancia en que se colocó las radios.

Las condiciones en que se realizó el experimento fueron las siguientes:
Los datos se tomaron dentro de un domicilio, en donde las mediciones obtenidas a 1 y 3 metros fueron a línea vista y las mediciones obtenidas a 6, 15, 20 y 25 metros fueron entre paredes de concreto. Las diferentes frecuencias de transmisión a las que se enviaron los datos fueron de 1, 3, 5 y 10 segundos. En la aplicación el tiempo de espera para leer un dato fue de 1 segundo. A continuación explicaremos el experimento realizado:

En la figura 4.13 se pudo observar que la perdida de paquetes se incrementó conforme la distancia aumentó. Adicionalmente se pudo notar que a los 20 metros y a una frecuencia de transmisión de 1 segundo hubo una perdida mayor de paquetes, esto pudo haberse dado debido a la ubicación del receptor en ese momento, ya que realizamos las pruebas dentro de una casa.

La curva estadística de 3 segundos nos muestra que a los 20 metros se presentó más perdida de tramas que a una distancia mayor, esto puede haberse dado debido a que el lugar donde se ubicó el teléfono y la radio fue diferente que al resto de otras transmisiones.

Las curvas estadísticas de 10 y 5 segundos presentaron un comportamiento proporcional entre el número de tramas perdidas y la distancia entre las radios. Finalmente podemos observar que la perdida de paquetes a una frecuencia de 10 segundos es muy baja llegando a un máximo de 10%.

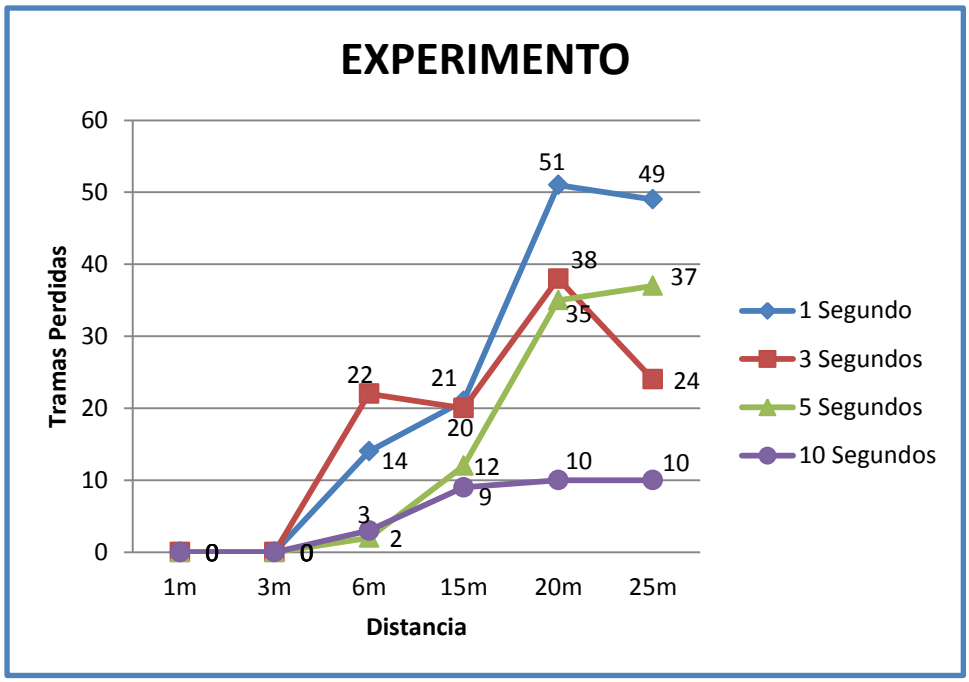

Figura 4.13.- Perdida de paquetes a diferentes frecuencias

4.6 Conclusiones

Podemos concluir en base a los datos obtenidos, que la perdida de datos es proporcional a la distancia, y las variaciones de datos a distancia de 20 metros pudo haberse dado por el lugar donde se colocó el teléfono y por ende la radio, se debe tener en cuenta que las pruebas se realizaron dentro de una casa. También se observa que a mayor frecuencia existe menos perdida de datos y a distancia menores a 6 metros la perdida de datos en nula.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

En esta investigación se trató un problema planteado que es el no poder tomar datos en lugares de difícil acceso por el ser humano, mediante el planteamiento de una solución, un diseño inicial que fracaso debido a problemas de bloqueo y un diseño final que resolvió el problema encontrado en el primer diseño, se habilito el módulo FTDI en el sistema operativo Android con el fin de desarrollar una aplicación que tome datos mediante una radio, y se encontraron las siguientes conclusiones:

1. Las pruebas fueron realizadas en un ambiente interior, y como era de esperar, la interferencia de equipos electrónicos (televisores, teléfono inalámbrico y radio) afectó la potencia de la señal emitida por la radio conectada al computador; esto se dedujo debido a que al momento de

realizar una prueba de envió de datos en un laboratorio donde la única interferencia pudo provenir de la red inalámbrica (WIFI), a una distancia aproximada de 6 metros entre las radios, no se presentó perdida de información alguna. La pérdida de datos a distancias mayores a 30 metros fue tan grande que se decidió descartar los datos obtenidos durante los experimentos a estas distancias, ya que no representaban el real funcionamiento de las radios.

2. Después de realizar las pruebas en un ambiente cerrado, y analizar la información obtenida en los experimentos, se determinó que se perdió el 34% de las tramas en promedio, de las 100 enviadas, si nos basamos en los picos de cada una de las curvas. En comparación al promedio de perdida de paquete proporcionados por las especificaciones de las radios utilizadas, que es de alrededor del 1%, el rendimiento de las radios no es el esperado, pero se debe recordar que el ambiente de las pruebas no era el óptimo, debido a la interferencia y obstáculos presentes durante el experimento.

3. La creación de un hilo a nivel del núcleo del sistema operativo Android, resolvió el problema del bloqueo de la interfaz de usuario que se tuvo en el primer diseño de la aplicación y por ende la obtención de tramas, ya que ahora la aplicación se encarga de procesar la trama obtenida y de seguir leyendo datos del puerto de manera simultánea. En caso de que el teléfono

tuviera un procesador de múltiples núcleos, el procesamiento en paralelo sería posible, lo que mejoraría aún más la obtención de tramas a frecuencias más altas de envió de datos.

Recomendaciones

1. Habilitar el módulo FTDI fue la parte más complicada en el desarrollo de este trabajo ya que se tuvo que buscar el núcleo apropiado para el tipo de teléfono que se tenía, y una vez encontrado había que configurarlo de tal que se pueda instalar en el teléfono sin que se produzca ningún error.

2. Para poder tener acceso a un dispositivo mediante el módulo FTDI no basta con instalar este módulo, también se debe revisar que la versión del núcleo de Android que se tiene tenga soporte OTG es decir se pueda utilizar el teléfono como un anfitrión.

3. El desarrollo de la aplicación inicialmente fue considerado para otras radios, las cuales no estuvieron bajo nuestra responsabilidad, lo que implicó que se realicen experimentos y cambios en el funcionamiento de las mismas. Lógicamente esto afectó la ejecución de nuestra aplicación, lo cual represento pérdida de tiempo en determinar el problema y solucionarlo. Se debe trabajar con recursos dedicados, o que no cambien su funcionamiento básico.

4. Las limitaciones del hardware de los teléfonos actuales deben ser revisadas antes de querer modificar el núcleo de un sistema operativo. A pesar de que se indicaba que la versión utilizada en nuestro teléfono de prueba inicial soportaba estos cambios, las limitantes del hardware impidieron su funcionamiento.

5. Debemos tener cuidado al modificar el núcleo de Android con Odín, ya que si se interrumpe el proceso de modificación ya sea porque el teléfono se descargue o se desconecte, dicho teléfono quedara inutilizable.

ANEXOS

Instalación del Kit de Desarrollo Java (JDK)

Instalar JDK, existen varios pasos que debemos realizar para una instalación exitosa:

Tenemos que bajar la versión deseada de Eclipse desde la página http://www.oracle.com/technetwork/java/javase/downloads/index.html

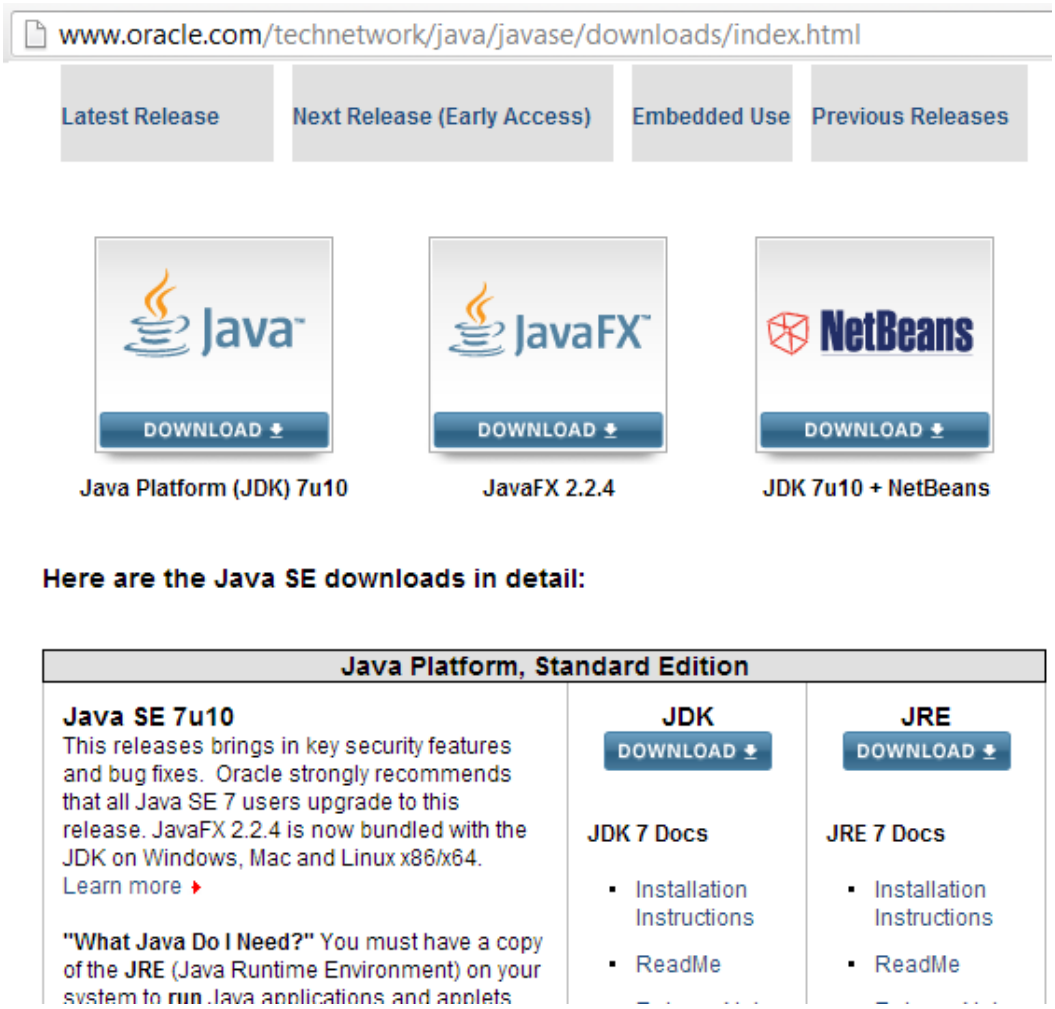

Figura 1.- Descarga de Eclipse

Java SE Development Kit 7u10

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux x86	106.63 MB	jdk-7u10-linux-i586.rpm
Linux x86	92.97 MB	jdk-7u10-linux-i586.tar.gz
Linux x64	104.75 MB	jdk-7u10-linux-x64.rpm
Linux x64	91.71 MB	jdk-7u10-linux-x64.tar.gz
Mac OS X x64	143.46 MB	jdk-7u10-macosx-x64.dmg
Solaris x86 (SVR4 package)	135.61 MB	jdk-7u10-solaris-i586.tar.Z
Solaris x86	91.97 MB	jdk-7u10-solaris-i586.tar.gz
Solaris SPARC (SVR4 package)	135.79 MB	jdk-7u10-solaris-sparc.tar.Z
Solaris SPARC	95.3 MB	jdk-7u10-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	22.86 MB	jdk-7u10-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	17.57 MB	jdk-7u10-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	22.64 MB	jdk-7u10-solaris-x64.tar.Z
Solaris x64	15.02 MB	jdk-7u10-solaris-x64.tar.gz
Windows x86	88.72 MB	jdk-7u10-windows-i586.exe
Windows x64	90.36 MB	jdk-7u10-windows-x64.exe
Linux ARM v6/v7 Soft Float ABI	65.07 MB	jdk-7u10-linux-arm-sfp.tar.gz

Figura 2.- Descarga de JDK

Una vez realizada la descarga debemos elegir el directorio donde lo descomprimiremos, este va a ser /usr/lib/jvm

Los comandos para descomprimirlo serán:

tar zxvf jdk-<versión>-linux-i586.tar.gz (versión de 32 bits)

tar zxvf jdk-<versión>-linux-x64.tar.gz (versión de 64 bits)

Ahora procedemos a configurar el sistema operativo para que use la versión que acabamos de instalar.

sudo update-alternatives --install "/usr/bin/java" "java" "/usr/lib/jvm/jdk1.7.0/bin/java" 1

sudo update-alternatives --install "/usr/bin/javac" "javac" "/usr/lib/jvm/jdk1.7.0/bin/javac" 1

sudo update-alternatives --install "/usr/bin/javaws" "javaws" "/usr/lib/jvm/jdk1.7.0/bin/javaws" 1

Y ahora ejecutamos

sudo update-alternatives --config java

Con lo cual obtendremos una pantalla indicándonos las opciones de java disponibles, escribimos el número de opción correspondiente a nuestra instalación.

Chequeamos nuestra versión

java –version

Y repetimos lo mismo con los comandos:

sudo update-alternatives --config javac

sudo update-alternatives --config javaws

Preparación del Entorno de Desarrollo

Introducción

Dependiendo del Sistema Operativo sobre el cual deseemos desarrollar para Android, deberemos seguir ciertos pasos que están indicados en la página de desarrolladores de google, (http://developer.android.com/sdk/installing/index.html). En nuestro caso en particular decidimos instalarlo en Ubuntu 11.10 de 32 bits, ya que al estar basado en Linux, sus herramientas de compilación y ejecución son más naturales de usar

En este apartado vamos a escribir lo pasos básicos para preparar nuestra computadora para el desarrollo de aplicaciones en Android.,

Para desarrollar aplicaciones en Android, podemos usar cualquiera de las siguientes plataformas:

Microsoft PC

Apple Mac OS X

Linux PC

Windows 7, Vista, Mac OS X y sistemas Linux son soportados en versiones de 32 y 64 bits, pero Windows XP solo en 32 bits.

Para comenzar nuestro desarrollo necesitamos software dedicado para el desarrollo Android:

Kit de Desarrollo Java (JDK)

Android SDK

Android NDK

Un ambiente integrado de desarrollo (Eclipse)

En nuestro caso particular, ya que Android está basado en Linux, sus herramientas de compilación y ejecución son más naturales de usar sobre una distribución Linux, nosotros hemos elegido utilizar Ubuntu 11.10 de 32 bits ya que es el ambiente con el que más estamos familiarizados.

Para trabajar con Android NDK vamos a tener que instalar y configurar algunos paquetes del sistema y utilidades, entre ellos:

Glibc.- Una herramienta para la generación de ejecutables, en este caso make que lo obtenemos al instalar los paquetes build-essential.

El JDK (Kit de Desarrollo Java).- que es un software que provee herramientas de desarrollo para la creación de programas en Java, para su correcta instalación existen varios pasos que debemos realizar, dichos pasos pueden ser vistos en el anexo 1.

Para compilar proyectos desde la línea de comandos, Linux soporta Ant, una utilidad de construcción automatizada basada en Java.

En estos momentos tenemos los paquetes y utilidades necesarios para comenzar a desarrollar en Android SDK y Android NDK.

Instalando Kit de Desarrollo de Software Android (SDK) y Kit de Desarrollo Nativo Android (NDK)

La instalación del SDK y NDK de Android es relativamente sencilla, aunque si toma un poco de tiempo dependiendo la velocidad de descarga que uno posea.

A continuación vamos a explicar los pasos necesarios para construir nuestro entorno de desarrollo:

Abrir un navegador e ir a http://developer.android.com/sdk. Seleccionamos el SDK disponible para nuestra plataforma (android-sdk_r21.0.1-linux.tgz) y lo descargamos en nuestro computador.

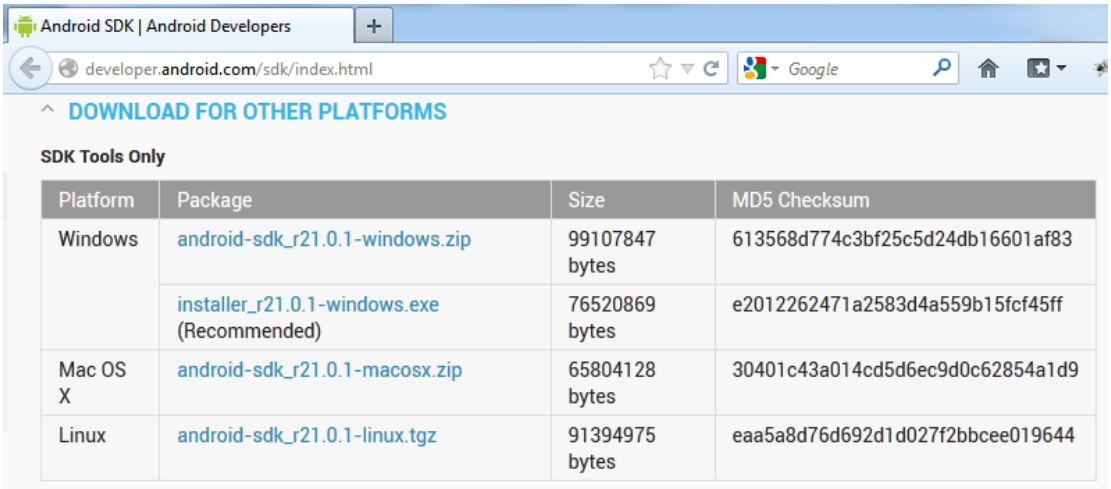

Figura 3. - Descarga de Kit de Desarrollo de Software Android (SDK)

Luego ir a http://developer.android.com/sdk/ndk y descargar el Android NDK para Linux.

Descomprimir los archivos descargados en los directorios de su elección, para nuestro caso lo hicimos en /home/<usuario>/Android/
En este caso nos quedaría de la siguiente manera:
/home/<usuario>/Android/SDK
/home/<usuario>/Android/NDK

Declaremos dos variables de entorno para referirnos a estos directorios
$ANDROID_SDK y $ANDROID_NDK
Lo hacemos haciendo modificando el archivo .profile que se encuentra en /home/<su usuario>, y agregando las siguientes líneas
export ANDROID_SDK=”<ruta del directorio Android SDK>”
export ANDROID_NDK=”< ruta del directorio Android NDK>”
export
PATH=”$PATH:$ANDROID_SDK/tools:$ANDROID_SDK/platformtools:$
ANDROID_NDK”

Grabar los cambios y salimos de nuestra sesión actual, esto para que se carguen las variables de entorno configuradas en el paso anterior.

Ingresar nuevamente y abrir un terminal para ingresar el comando: Android

Con esto la ventana del Android SDK y AVD Manager se mostraran

Ir a la sección de paquetes instalados y actualizarlos todos.

Una ventana aparecerá, hay que seleccionarlos todos e instalarlos.

Después de instalar todos los paquetes un mensaje solicitando reiniciar el servicio ADB aparecerá y presionamos el botón “yes”.

Ahora puede cerrar esta ventana del Android SDK y AVD Manager.

Bien, ya tenemos instalado los archivos del Android SDK y del Android NDK en nuestra máquina, pero ¿Cómo empezamos con la programación?

Exacto, configurando nuestro IDE Eclipse.

Instalación de Eclipse

Actualmente se puede desarrollar aplicaciones para Android haciendo uso de marcos de trabajo proporcionados por ciertos fabricantes e incluso existen aplicaciones que nos permite desarrollar directamente en el teléfono. ¿Por qué nuestra elección de Eclipse? Existen unas pocas razones para nuestra elección:

Manteniendo la línea de la Open Handset Alliance, de una real apertura en el mercado del desarrollo de móviles, Eclipse es uno de los IDE Java más completos y libres. Además es muy fácil de usar, con una curva mínima de

aprendizaje. Lo que convierte a Eclipse en un IDE muy atractivo para el desarrollo en Java.

La misma Open Handset Alliance ha liberado un complemento para Eclipse que permite crear proyectos específicos para Android, compilarlos, y usar el Emulador Android para correrlos y depurarlos. Adicionalmente este complemento configura ciertos elementos (como archivos y variables) por ti. Todo esto ayuda a no desperdiciar tiempo valioso de desarrollo y reduce la curva de aprendizaje, lo que significa que se puede utilizar más tiempo en el desarrollo de aplicaciones increíbles.

Después de descargar de la página http://www.eclipse.org/downloads/ (Eclipse IDE para desarrolladores java es suficiente para nuestro desarrollo)
Extraer el archive descargado Tar/Gz
Una vez extraído, corremos Eclipse dando doble click sobre el icono ejecutable dentro del directorio.
Si Eclipse pregunta por un directorio de trabajo (workspace), defina el que usted desee y de en Ok.
Después de que Eclipse haya comenzado cierre la pantalla de bienvenida.
Vaya al menú Ayuda => instalar nuevo software

Seleccionamos agregar y en la ventana que nos aparece ponemos como nombre Android y como dirección ingresamos https://dl-ssl.google.com/android/eclipse/ y damos aceptar.

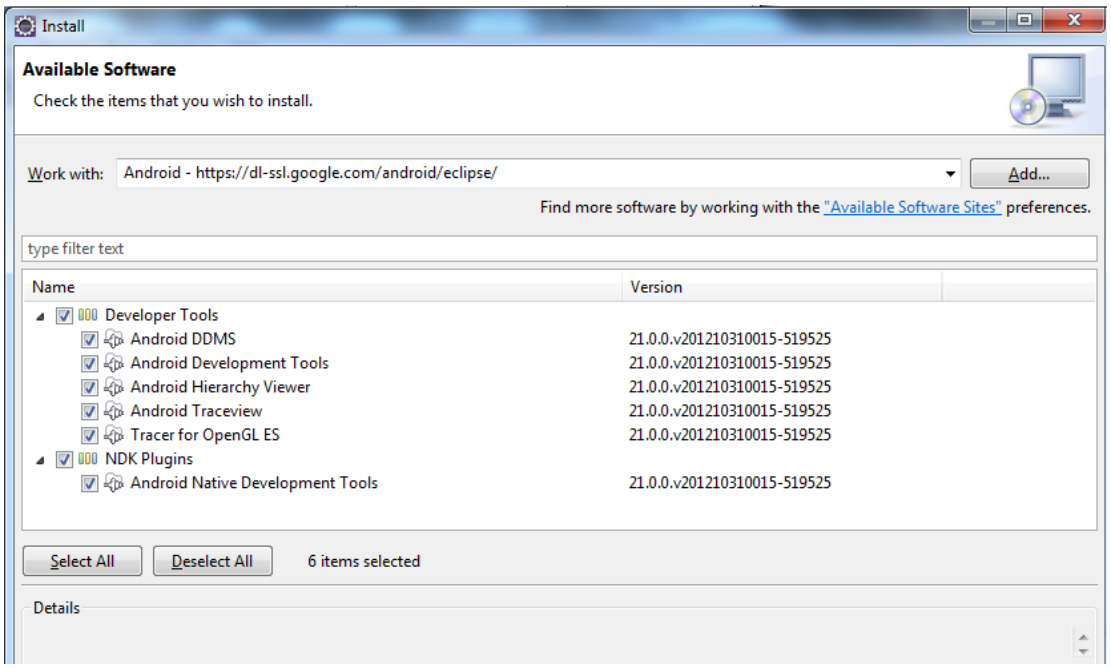

Figura 4.- Habilitar Software en SDK

Después de unos segundos aparecerá el complemento de Herramientas de Desarrollo, hay que seleccionarlo y dar click en siguiente.

Seguir el asistente para aceptar las condiciones cuando lo pregunten.

Una vez que se instale el ADT, una advertencia de que el complemento no está firmado digitalmente aparecerá, ignórelo y de click en Ok.

Cuando termine totalmente la instalación, solicitaran reiniciar eclipse, es necesario hacerlo para que los cambios se apliquen correctamente.

Después de hacerlo vaya a opción Windows => Preferencias y vaya a la sección Android

Click en buscar y seleccione el directorio donde está instalado Android SDK

Le aparecerá un listado de los API de Android instalados, de click en Aceptar

Vaya a ayuda, instalar nuevo software

Abra la lista que dice Trabajar con… y seleccione el ítem que contiene la versión de Eclipse correcta.

Encuentre lenguajes de programación en el árbol de plugins que le aparece.

Seleccione CDT plugins y C/C++ Call Graph Visualization

Siga el asistente y acepte las condiciones.

Una vez instalado, reinicie Eclipse

Existe un complemento que puede ser usado para poder escribir parte de tu aplicación Android usando código nativo. Para nuestro caso no es necesario instalarlo ya que vamos a compilar y ejecutar nuestra aplicación desde consola.

Emulando Android

Android SDK provee un emulador para ayudar a los desarrolladores a probar sus aplicaciones antes de tener un dispositivo con Android en sus manos.

Los pasos para crear un Dispositivo Virtual Android (AVD) son los siguientes:

Abrir el Manejador AVD desde el menú Windows en Eclipse.

Click en nuevo

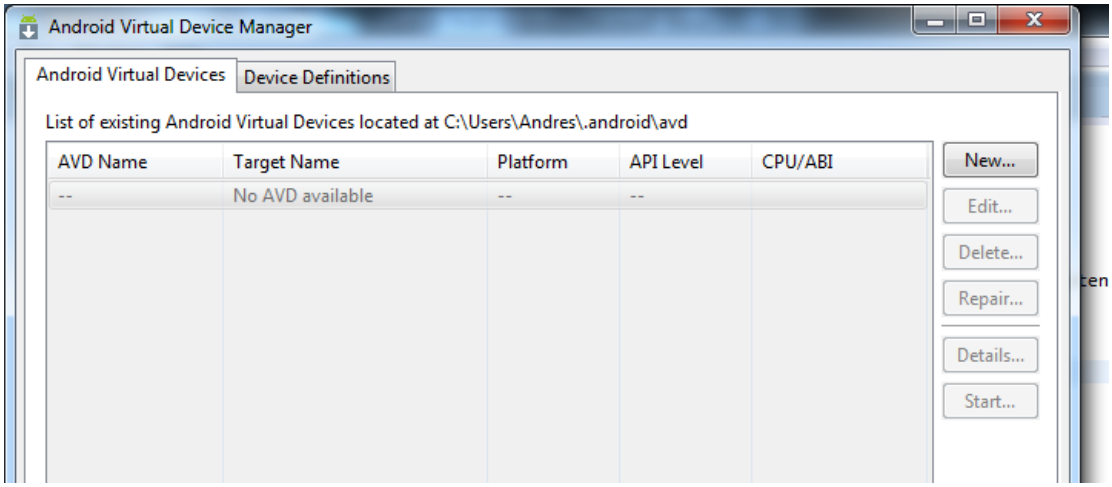

Figura 5.- Administrador de Dispositivos Virtuales de Android

Dar un nombre a este emulador

Elegir la versión de Android que se desee

Especificar las características deseadas, como son el tipo de CPU, el almacenamiento interno, cámaras frontal y posterior en caso de necesitarlas

Dar click en crear AVD

Con esto el nuevo dispositivo aparece en la lista del Manejador AVD. Ahora veamos cómo funciona, seleccionemos de la lista el dispositivo virtual creado, y demos click en el botón iniciar, en la ventana que aparece seleccionamos arrancar. Unos pocos segundos después su dispositivo aparecerá de la siguiente manera

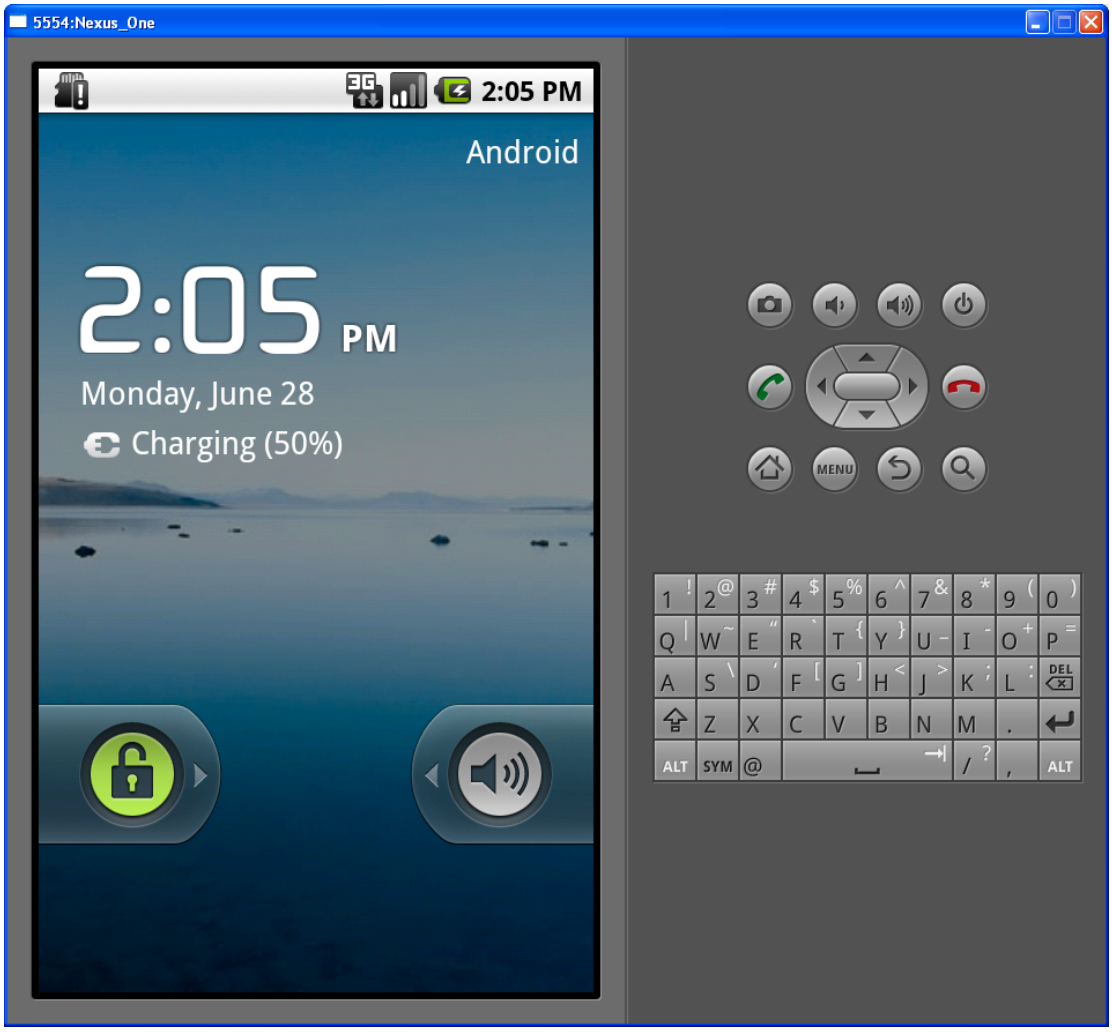

Figura 6.- Emulador de Android

Configuración de nuestro dispositivo Android

Para nuestra demostración, debemos conectar nuestro dispositivo Android a Ubuntu, por lo que necesitaremos seguir los siguientes pasos:

En su dispositivo Android hay que habilitar las opciones de desarrollo, esto puede variar entre dispositivos, esto usualmente está dentro del menú del sistema, la opción de Aplicaciones

Una vez allí habilite la depuración USB y seleccione Permanecer Encendido

Conecte su dispositivo a su computadora usando un cable de datos. Dependiendo de su dispositivo, este aparecerá como un disco USB.

Intente correr ADB y liste los dispositivos, esto lo hacemos desde consola ejecutando el comando

adb devices

Si apareciera ????????? en lugar del nombre de su dispositivo, estos significa que no tiene los accesos apropiados. Hay que encontrar su vendedor id y producto id. El vendedor ID es un valor fijo para cada vendedor, en la siguiente lista podemos ver los proporcionados en la página: http://developer.android.com/tools/device.html#VendorIds

Company	**USB Vendor ID**
Acer	0502
ASUS	0b05
Dell	413c
Foxconn	0489
Fujitsu	04c5
Fujitsu Toshiba	04c5
Garmin-Asus	091e
Google	18d1
Hisense	109b
HTC	0bb4
Huawei	12d1
K-Touch	24e3
KT Tech	2116

Kyocera	0482
Lenovo	17ef
LG	1004
Motorola	22b8
NEC	0409
Nook	2080
Nvidia	0955
OTGV	2257
Pantech	10a9
Pegatron	1d4d
Philips	0471
PMC-Sierra	04da
Qualcomm	05c6
SK Telesys	1f53
Samsung	04e8
Sharp	04dd
Sony	054c
Sony Ericsson	0fce
Teleepoch	2340
Toshiba	0930
ZTE	19d2

Tabla 1.- Lista de Compañías

El producto id puede ser encontrado usando el comando lsusb y filtrado con el id del vendedor

lsusb | grep <id_vendedor>

Con el usuario root crear un archivo /etc/udev/rules.d/52-android.rules con Vendedor y Producto ID:

sudo sh -c ‘echo SUBSYSTEM==\”usb\”, SYSFS{idVendor}==\”<Your

Vendor ID>\", ATTRS{idProduct}=\"<Your Product ID>\",

MODE=\"0666\" > /etc/udev/rules.d/52-android.rules'

Hay que cambiar los permisos del archivo a 644, lo que corresponde a permisos de lectura para el propietario, el grupo y cualquier otro usuario, y de escritura solo para el usuario.

sudo chmod 644 /etc/udev/rules.d/52-android.rules

Reiniciar el servicio udev, que es el manejador de dispositivos del núcleo de Linux

sudo service udev restart

Relanzar el servidor ADB como administrador

sudo $ANDROID_SDK/tools/adb kill-server

sudo $ANDROID_SDK/tools/adb start-server

Hay que verificar que aparezca el dispositivo usando el comando

adb devices

Si no aparece, debió existir algún error en los pasos anteriores.

Superusuario en Android

Para muchas tareas en Linux, necesita autoridad de root o de superusuario. El usuario root, algunas veces llamado el superusuario, es el usuario que normalmente es usado para tareas administrativas como configurar el

sistema o instalar software. Use superusuario sólo cuando necesite realizar tareas administrativas; evite usar superusuario para su trabajo normal. El usuario superusuario puede hacer cualquier cosa, incluyendo destruir accidentalmente su sistema, lo no normalmente no es algo bueno. Los usuarios tienen menos privilegios y el sistema está mucho más protegido de ser inadvertidamente dañado por usuarios normales.

Al igual que Linux, Android necesita que el usuario tenga privilegios de superusuario para poder modificar archivos de su núcleo, instalar módulos e instalar roms.

Los fabricantes de celulares bloquean esta opción de usuario superusuario, para que los usuarios de los teléfonos celulares no tengas acceso a los archivos del sistema y puedan dañar el núcleo y también para evitar que los usuarios instales aplicaciones pagadas de manera gratuita.
Una vez que procedemos a volver superusuario el Teléfono en la lista de aplicaciones aparece la aplicación “Superusuario”, que nos indica que el teléfono está ahora en modo superusuario.

ROM

La ROM es conocida como la memoria de “Solo Lectura” pero con el pasar del tiempo ha cambiado su uso. Ahora también puede considerarse como

una zona no volátil de la Memoria Flash.

En Android la ROM es un archivo en formato zip que guarda la configuración del sistema o el programa de arranque del teléfono.

Modo de Recuperación

Es un modo alterno del Teléfono, en este modo podemos instalar Roms, hacer Wipe Data, Wipe Cache Partition, Wipe Dalvik Cache, realizar copias de seguridad del sistema y permite restaurar estas copias del sistema.

El modo de recuperación de algunos teléfonos solo permite instalar Rom Oficiales y no Custom Roms, para solucionar esto podemos instalar algunos de los modos de recuperación que se encuentran en formato zip, la más utilizada es ClockworkMod.

Comandos ADB

Esta es una pequeña lista de algunos comandos sencillos que pueden usar en ADB

- adb shell - inicia una conexión de shell con el teléfono
- adb push - envía un archivo al teléfono a través de ADB a través de USB -ejemplo- adb push c:\test.apk/sdcard/test.apk
- adb pull - recibe un archivo desde el teléfono a través de ADB a través de USB -ejemplo- adb pull /system/app/Test.apk c:\Test.apk

- adb reboot - reiniciar el teléfono
- adb reboot recovery - se reinicia el teléfono en modo recuperación (xRecovery)
- adb reboot bootloader - reinicia el teléfono en el cargador de arranque (pantalla en blanco)
- adb remount - vuelve a montar el sistema de archivos
- adb install - instala una aplicación -ejemplo- instalar adb c:\swype.apk

BIBLIOGRAFIA

[1] Olivas Amaya Luis Adrián, “Analizar e identificar las tecnologías de comunicación inalámbricas, su aplicación y usos dentro del Instituto Tecnológico de Durango”, http://es.scribd.com/doc/36485035/Tecnologias-de-comunicacion-inalambricas, Junio del 2010

[2] Alejandro Nieto Gonzalez, “¿Qué es Android?”, http://www.xatakandroid.com/sistema-operativo/que-es-android, Fecha de Publicación: 8 de Febrero del 2011

[3] Blanco Lezcano y Jent Chong, Facultad de Electrotecnia y Computación Universidad Nacional de Ingeniería, “Android Operating System”, http://electrouni.files.wordpress.com/2010/12/android-os.pdf,
16 de Noviembre del 2010

[4] Bill Anderson, “Android es más que otra distribución de Linux” http://www.all-things-android.com/es/content/android-es-m%C3%A1s-que-otra-distribuci%C3%B3n-de-linux, Fecha de Publicación: 26 de Julio del 2012

[5] comScore, “comScore Reports June 2013 U.S. Smartphone Subscriber Market Share”,

http://www.comscore.com/Insights/Press_Releases/2013/8/comScore_Reports_June_2013_U.S._Smartphone_Subscriber_Market_Share, Fecha de Publicación: 7 de Agosto del 2013

[6] Paul Paliath, “Android 4.1 JellyBean vs iOS 6 vs Windows Phone 8 – TheUltimateComparison”, http://www.redmondpie.com/android-4.1-jelly-bean-vs-ios-6-vs-windows-phone-8-the-ultimate-comparison/, Fecha de Consulta: 6 de Julio del 2012

[7] El Androide Libre, “8 Razones por las que Android es mejor que iPhone”, http://www.elandroidelibre.com/2010/01/8-razones-por-las-que-android-es-mejor.html, Fecha de consulta: 30 de Marzo del 2011

[8] Mobileburn, “What is NFC?”, http://www.mobileburn.com/definition.jsp?term=NFC, 12 Junio del 2013

[9] Margaret Rouse, “virtual machine (VM)”, http://searchservervirtualization.techtarget.com/definition/virtual-machine, 31 de Octubre del 2011

[10] UTPL, jfgalvez, “Máquinas Virtuales” http://blogs.utpl.edu.ec/sistemasoperativos/2010/01/18/maquinas-virtuales-6/,

18 de Enero del 2010

[11] Condesa, “La máquina virtual Dalvik”, http://androideity.com/2011/07/07/la-maquina-virtual-dalvik/, Fecha de publicación: Julio 7 del 2011

[12] AndroidDevelopers, “Glosario Android”, http://developer.android.com/guide/appendix/glossary.html, Fecha de consulta: Enero 2013

[13] David Ehringer, “The Dalvik Virtual Machine Arquitecture”, http://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf, Marzo, 2010

[14] Android: a programmer’s guide; Jerome DiMarzio; McGraw-Hill; 2008;

[15] Android Architecture, http://elinux.org/Android_Architecture, Fecha de Consulta: Junio del 2013

[16] androideity, “Arquitectura de android”, http://androideity.com/2011/07/04/arquitectura-de-android/, Fecha de publicación: Julio 4 del 2011

[17] Hello, Android: Introducing Google's Mobile Development Platform; Ed Burnette; Pragmatic Bookshelf; 2010;

[18] Linux Wiki, "Android Kernel Features", http://elinux.org/Android_Kernel_Features, Fecha de publicación: 12 de Diciembre del 2011

[19] Android Developers, "Contents of the NDK", http://developer.android.com/tools/sdk/ndk/index.html, Fecha de consulta: 1 de mayo del 2013

[20] Linaro, "Android OS for Servers?", http://elinux.org/images/8/89/Elc2011_stultz.pdf, 13 de Abril del 2011

[21] Xianzhong Zhu, "Introduction to Android JNI development Using NDK", http://dotnetslackers.com/articles/net/Introduction-to-Android-JNI-development-Using-NDK-Part-1.aspx, Fecha de Publicación: 16 de diciembre del 2011

[22] Android Developers, "Glosario Android", http://developer.android.com/guide/appendix/glossary.html, Fecha de consulta: Enero 2013

[23] FutureTechnologyDevices International Limited, “Opciones para periféricos en Android”, http://www.ftdichip.com/Support/Documents/White_Papers/WP_003_Android_Peripheral_Options.pdf, 11 Febrero del 2013

[24] AndroidDevelopers, “Fundamentos Android”, http://developer.android.com/guide/components/fundamentals.html, Fecha de consulta: Enero 2013

[25]Ian Shields, Senior Programmer IBM, “Volviéndose un superusuario”, http://www.ibm.com/developerworks/ssa/linux/tutorials/l-basics/section5.html, Fecha de Publicación: 23 Abril 2012

[26] Adrian Latorre “USB On-The-Go (USB OTG)”, http://www.elandroidelibre.com/2012/06/conecta-cualquier-usb-a-tu-android-con-usb-on-the-go.html, Fecha de consulta: abril del 2013

[27] XDA-University, “Android Dictionary”, http://xda-university.com/as-a-user/android-dictionary, Fecha de Consulta: 30 de Enero del 2013