ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

"LOCALIZACIÓN DE PERSONAS EN INTERIORES POR MEDIO DE REDES WiFi USANDO EL SISTEMA OPERATIVO ANDROID"

TESINA DE SEMINARIO

Previa a la obtención del Título de:

INGENIERO EN CIENCIAS COMPUTACIONALES

INGENIERO EN TELEMÁTICA

Presentado por

DANIEL ANDRÉS TIGSE PALMA

CATALINA ISABEL SOLÍS MIRANDA

GUAYAQUIL - ECUADOR

2013

AGRADECIMIENTO

Agradezco a Dios por haber querido que culmine una etapa más de mi vida, a mi familia por apoyarme en todo momento, a mis amigos que en el camino conocí.

Daniel Tigse Palma.

Agradezco a Dios por todas las bendiciones a lo largo de esta etapa académica, a mi familia y a todas las personas involucradas en mi desarrollo profesional.

Catalina Solís Miranda.

DEDICATORIA

Dedico este trabajo a mis padres por su apoyo incondicional.

Daniel Tigse Palma.

Dedico este trabajo a mis hermanas ya que ellas han sido mi constante apoyo, a mis padres en el cielo.

Catalina Solís Miranda.

TRIBUNAL DE SUSTENTACIÓN

Dr. Daniel Ochoa Donoso, PH.D.

PROFESOR DEL SEMINARIO DE GRADUACIÓN

Ing. Ignacio Marín García, MSIS

PROFESOR DELEGADO POR EL DECANO DE LA FACULTAD

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de este Trabajo de Graduación, nos corresponde exclusivamente; y el patrimonio intelectual de la misma, a la Escuela Superior Politécnica del Litoral"

(Art. 12 Reglamento de Graduación de la ESPOL)

Daniel Tigse Palma.	**Catalina Solís Miranda.**

RESUMEN

El presente trabajo provee una forma práctica de cómo aplicar conceptos de procesamiento concurrente como son el uso de hilos y la sincronización y comunicación entre procesos en dispositivos móviles. Como implementación de estos conocimientos adquiridos se realizó un proyecto que consiste en el desarrollo de una aplicación móvil para el sistema operativo Android en donde el usuario en una de las opciones de la aplicación guarda una lista de artículos que desee comprar en un centro comercial. Nuestra aplicación dio aviso al usuario mediante una notificación cuando este pasaba cerca de un local donde se comercializaban alguno de los artículos que se encontraban en esta lista. La aplicación fue programada en lenguaje C para el procesamiento de datos y llamadas al sistema; solo se empleó lenguaje Java para el acceso a sensores del teléfono e interfaz gráfica.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO

Audio Flinger: Es el componente del Android que administra el audio en espacio de usuario antes de enviarlo al controlador del núcleo.

Identificador de conjunto de servicios: Del inglés 'Basic Service Set Identifier', es un nombre de identificación único de todos los paquetes de una red inalámbrica (Wi-Fi) para identificarlos como parte de esa red.

Interfaz de programación de aplicaciones: Del inglés 'Application Programming Interface', es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Kit de desarrollo nativo: El NDK o 'Native Development Kit' por sus siglas en inglés es un conjunto de herramientas que permite incorporar componentes para hacer uso de código nativo en aplicaciones de Android.

Kit de desarrollo de plataforma: El Android PDK o 'Platform Development Kit', es un conjunto de herramientas para desarrolladores y fabricantes creado con el objetivo de combinar software y hardware al hacer accesorios para la plataforma Android.

Kit de desarrollo de software: Se refiere al kit de desarrollo de software que es un conjunto de herramientas que le permite al programador crear aplicaciones para un sistema concreto, por ejemplo ciertos paquetes de software, frameworks, computadoras, videoconsolas, sistemas operativos, etc.

Llamada a procedimiento remoto: Del inglés 'Remote Procedure Call', es un protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos. El protocolo es un gran avance sobre los sockets usados hasta el momento. Las RPC son muy utilizadas dentro del paradigma cliente-servidor.

Paquete de Android: Se refiere a la extensión (.apk) de un paquete para el sistema operativo Android. Este formato es una variante del formato JAR de Java utilizado para distribuir e instalar componentes empaquetados para la plataforma Android en teléfonos inteligentes y tablets.

Sistema de Posicionamiento Global: Del inglés 'Global Positioning System', es un sistema global de navegación por satélite que permite determinar en todo el mundo la posición de un objeto, una persona o un vehículo.

Surface Flinger: Es un componente nativo de Android que permite combinar superficies. Gracias a un buffer crea las superficies de las aplicaciones, puede combinar superficies 2D y 3D dependiendo del dispositivo, también puede usar OpenGL ES y acelerador de hardware 2D para sus composiciones.

WakeLocks: son una característica del sistema Power Manager Service, disponible para controlar el estado de alimentación del dispositivo host, pueden ser usados para mantener el funcionamiento del CPU, evitar que la pantalla se atenúe, evitar que la pantalla se apague, y otros propósitos que ayudan a la batería.

WIFI: es un mecanismo de conexión de dispositivos electrónicos de forma inalámbrica. Los dispositivos habilitados con WIFI, tales como: un ordenador personal, una consola de videojuegos, un Smartphone o un reproductor de audio digital, pueden conectarse a Internet a través de un punto de acceso de red inalámbrica.

ABREVIATURAS

API	Interfaz de programación de aplicaciones
APK	Paquete de Android
BSSID	Identificador de conjunto de servicios
GPS	Sistema de posicionamiento global
NDK	Kit de desarrollo nativo
PDK	Kit de desarrollo de plataforma
RPC	Llamada a procedimiento remoto
SDK	Kit de desarrollo de software

INTRODUCCIÓN

En la actualidad el uso de teléfonos inteligentes ha venido creciendo, de acuerdo con el Instituto Nacional de Estadísticas y Censos del Ecuador, en el año 2012, el 8.4% de las personas que poseen un celular tienen un teléfono inteligente, cifra que ha crecido en comparación con años anteriores [1]. Por ello, para algunos desarrolladores la nueva plataforma de trabajo es la móvil. También se puede notar que empresas como Banco del Pichincha o Farmacias Pharmacys, que su enfoque no es el campo tecnológico, no sólo tienen una página web sino ahora disponen de una aplicación móvil para mantener a sus usuarios más informados o para que accedan a los servicios que ofrecen.

El proyecto que desarrollamos consistió en crear un sistema capaz de identificar la posición del usuario con respecto a la posición de un local comercial, usando como fuente de información los datos que provee el GPS del teléfono móvil y los puntos de acceso WiFi. Los datos correspondientes a posiciones geográficas y niveles de potencias de señales WiFi son procesados utilizando técnicas de programación concurrente y manejo de múltiples hilos para que de manera más eficiente, es decir usando la menor cantidad de memoria y con el menor porcentaje de error, permitan determinar la posición del usuario y su proximidad a una tienda de interés. Nuestra idea fue plasmada en una aplicación para el sistema operativo Android, pero a diferencia de la mayoría de aplicaciones que utilizan el kit de desarrollo de software (SDK) con lenguaje de programación Java, nuestro proyecto fue desarrollado

utilizando el Kit de desarrollo nativo (NDK) que nos permite utilizar lenguaje C y explotar las llamadas al sistema operativo.

Evaluamos el comportamiento de las señales WiFi y las señales GPS en un entorno cerrado en donde existen muchas redes inalámbricas, muchas de estas con interferencia. Creamos criterios de búsqueda basados en pruebas que nos ayudaron a comprobar si con los datos de estas señales se podía determinar la posición de una persona con respecto a un local comercial.

La aplicación antes nombrada a la que llamamos GeoMemo ayudará a encontrar de manera más rápida cualquier tipo de artículo que se esté buscando dentro de un centro comercial, llamando la atención del usuario mediante una notificación cuando este camine cerca de la tienda en donde se encuentre el artículo deseado.

El contenido de nuestro informe está dividido en los siguientes capítulos. En el capítulo uno se describe la justificación del proyecto así como los objetivos generales y específicos.

En el capítulo dos se presenta un resumen sobre las tecnologías usadas, sus características y ventajas.

En el capítulo tres se explica el diseño del proyecto y los conceptos implementados para la solución del mismo.

Finalmente en el capítulo cuatro se muestran las pruebas realizadas desde el inicio de la implementación del proyecto.

CAPITULO 1

1. GENERALIDADES

En este capítulo se detalla la justificación del proyecto al igual que el objetivo general y los objetivos específicos.

1.1. Justificación

Las personas en la actualidad con los diferentes horarios y ritmo de vida que tienen tienden a olvidarse de adquirir artículos que necesitan como medicina, alimentos, vestimenta, entre otros. Nuestra aplicación GeoMemo resuelve este problema permitiendo a las personas guardar memos o recordatorios de artículos que venden dentro de un

centro comercial para que luego cuando estas estén cerca del local comercial donde vendan su artículo la aplicación les notifique.

1.2. Objetivo general

Estudiar y explotar las características y capacidades de un sistema operativo en un teléfono móvil mediante el acceso al hardware y uso de librerías para programación concurrente de procesos.

1.3. Objetivos específicos

- Estudiar las capacidades de un sistema operativo móvil en el uso del GPS e interfaces inalámbricas de red.
- Combinar arquitecturas para que varios procesos resuelvan una tarea de manera conjunta en un sistema operativo diseñado para un teléfono móvil.
- Diseñar e implementar un programa que explote las ventajas de los entornos de desarrollo para sistemas operativos móviles.

Estos objetivos los alcanzamos dentro del marco del desarrollo de una aplicación móvil para el sistema operativo Android, GeoMemo, que permita a los usuarios recordar artículos que deseen adquirir y

donde adquirirlos. Mediante el uso de sus coordenadas geográficas y las redes WiFi que se encuentren alrededor del local de interés.

1.4. Alcances y Limitaciones del Proyecto

- Demostrar que es posible crear una aplicación móvil utilizando dos marcos de trabajo (SDK y NDK) en conjunto con procesamiento concurrente.
- No pudo ser probado en un teléfono con gran capacidad de procesamiento debido a falta de recursos económicos.
- La aplicación solo fue probada dentro de un ambiente cerrado (C.C. Mall del sol).

CAPITULO 2

2. MARCO TEÓRICO

En este capítulo se encuentran resumidos los conceptos teóricos utilizados para la implementación de nuestro proyecto.

2.1. Android y Linux

Android, es un sistema operativo para dispositivos móviles de código abierto, basado en el núcleo de Linux [2] al cual se le han agregado diferentes librerías y funciones para optimizar su funcionamiento, el primer núcleo de Linux utilizado fue la versión 2.6.27 ya que era la tecnología de punta de esa época. A medida que se han dado las actualizaciones del núcleo de Linux, Google las ha utilizado para

realizar sus propios avances en el núcleo de Android obteniendo diferentes versiones hasta llegar a la actual versión 4.2 llamada Jelly Bean basada en la versión del núcleo de Linux 3.0.31 [3], tal como muestra la tabla I.

Tabla I. Versiones Kernel Android y Linux

Nombre	Versión	Kernel Linux
Cupcake	1.5	2.6.27
Donut	1.6	2.6.29
Eclair	2.0/2.1	2.6.29
Froyo	2.2	2.6.32
Gingerbread	2.3	2.6.35
HoneyComb	3.0/3.1/3.2	2.6.36
Ice cream Sandwich	4.0	3.0.1
Jelly Bean	4.1/4.2	3.0.31

Google añadió al núcleo de Linux diversas bibliotecas que se originan a partir de proyectos de código abierto con el fin de obtener una mayor funcionalidad. En el artículo llamado *Porting Android to a new Device- What did Google change in the Kernel* [4] se puede encontrar una lista completa de los cambios realizados al núcleo de Linux.

Los componentes más relevantes para nuestro proyecto que Google modificó son el controlador de alarma, Ashmem, Binder y la administración de energía. Los beneficios del ahorro de energía de suspender e hibernar provienen del hecho de que la mayor parte o todo el hardware está apagado, pero esto puede ser una limitación si

se espera algo de la funcionalidad del sistema, en especial para los teléfonos inteligentes los cuales tienen que activarse para hacer cosas como notificar al usuario de eventos de calendario o para comprobar si hay correo. El **controlador de alarma** le permite a un proceso del espacio de usuario (*user space*) notificar al núcleo cuando va a usar el procesador para ejecutar su código por medio de temporizadores físicos. Estos temporizadores utilizan un enfoque híbrido, cuando el sistema está en funcionamiento estos activan un temporizador de alta resolución para activarse cuando un evento se debe ejecutar, sin embargo, cuando el sistema entra en suspensión, el código de los temporizadores de alarma mira en la lista de eventos que existen y establece el RTC (Real time clock) para disparar una alarma cuando el primer evento que necesita ejecutarse lo realice [5].

Ashmem es un componente del sistema operativo Android que facilita el intercambio de la memoria entre los procesos. Proporciona un mecanismo para que el sistema operativo recupere memoria si este se encuentra con problemas de insuficiencia de la misma, lo que puede suceder fácilmente en un dispositivo con memoria limitada como es un teléfono móvil. Un proceso puede crear un área de memoria compartida nueva abriendo el archivo de dispositivo */dev/ashmem* y se obtiene el descriptor de archivo, luego llama al comando *ioctlashmem_set_name* para establecer el nombre a un dispositivo virtual y finalmente llama al comando

ioctlashmem_set_size para establecer el tamaño en bytes. El descriptor de archivo se puede compartir con otros procesos, utilizando otro servicio llamado *binder*, cada proceso mapea el archivo y se logra tener un bloque de memoria compartida.

Ashmem tiene algunas ventajas sobre otros métodos de memoria compartida, a diferencia del código de memoria compartida existente en el *Kernel* de Linux, la memoria se libera cuando el proceso termina. Un proceso podría asignar parte de la memoria como anónimo y luego realizar un fork del mismo pero ashmem es más flexible, ya que permite a un proceso decidir si comparte la memoria después de que haya realizado el fork. Ashmem ahorra memoria permitiendo a un proceso realizar la designación de algunas páginas de su memoria como disponibles. Ashmen también puede ser usada en otros sistemas embebidos [6].

Los servicios del sistema se proporcionan como un proceso servidor tal como se explica en el Anexo B, es necesario un mecanismo para enviar requerimientos y respuestas a procesos clientes, en Android este mecanismo se llama **Binder**. Por lo tanto, Android utiliza las funciones proporcionadas por otros procesos a través del Binder [Figura 2.1].

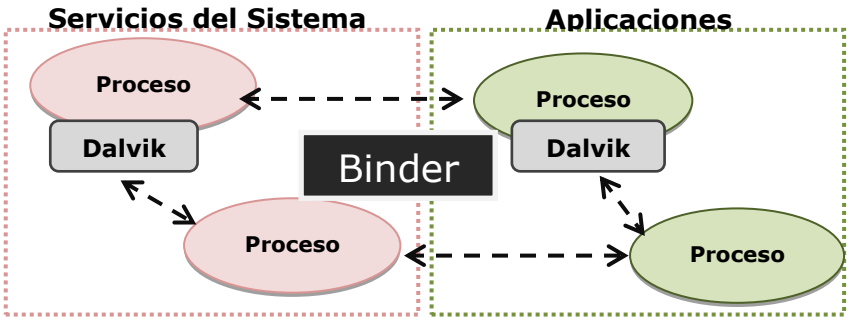


Figura 2.1. Binder, Comunicación entre Procesos.

La razón de haber desarrollado un nuevo mecanismo, en lugar de utilizar mecanismos de comunicación entre procesos (IPC), tales como *sockets* y *pipes* proporcionados por Linux fue porque mejora el rendimiento del núcleo. Binder hace referencia a la memoria compartida del *Kernel* para minimizar la sobrecarga causada por el copiado en memoria principal [Figura 2.2].

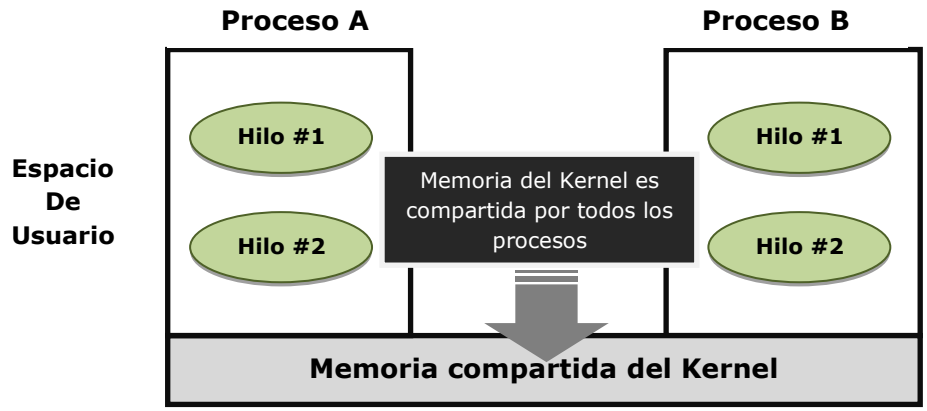


Figura 2.2. El espacio del núcleo compartido por todos los procesos.

El componente Binder consta de un *Driver* el cual se implementa como módulo del *kernel*. El papel del *Driver* es convertir la dirección

de memoria que cada proceso ha mapeado con la dirección de memoria del espacio del *Kernel* [Figura 2.3].

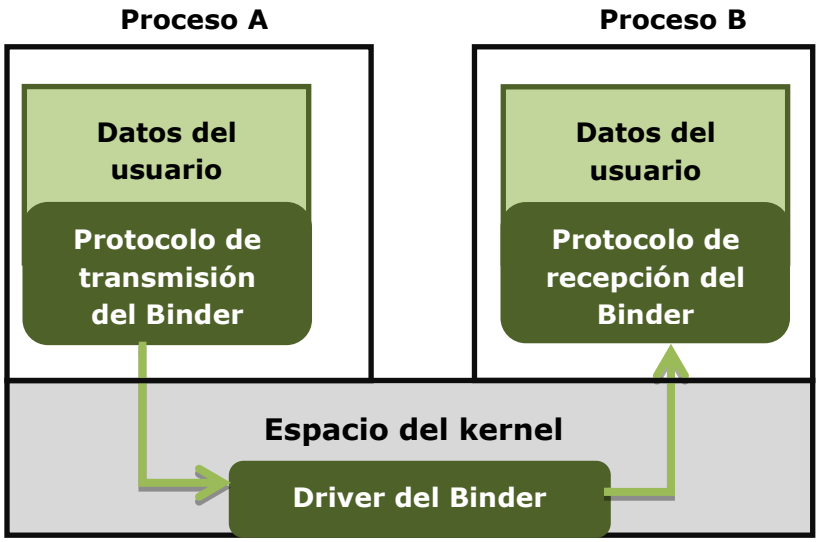


Figura 2.3. Configuración del controlador del Binder.

El mecanismo Binder parte de una idea simple: Dejar que los requerimientos y las respuestas sean escritos en una zona donde todos los procesos puedan compartir y dejar que cada proceso se haga referencia a la dirección de memoria correspondiente.

Para la computadora de escritorio normal, la **administración de energía** se utiliza para reducir el consumo de la misma y reducir las necesidades de enfriamiento. Menor consumo de energía significa menor necesidad de disipación de calor, ahorra dinero, reduce el impacto sobre el medio ambiente y aumenta la estabilidad del

sistema. Para los dispositivos móviles es mucho más importante porque la energía de la batería es muy limitada [8].

La administración de energía de Linux está diseñada para computadoras personales, en las cuales existen estados de ahorro de energía tales como suspender e hibernación. Sin embargo, estos mecanismos no son aplicables para dispositivos móviles. Por lo tanto, Android tiene una metodología adicional para el ahorro de energía. El administrador de energía de Android monitorea la vida de la batería, el estado del dispositivo, y lo apaga cuando alcanza un nivel insuficiente de energía. Las aplicaciones y servicios deben solicitar recursos del CPU con *wakelocks* usando el *framework* de aplicaciones de Android y librerías nativas de Linux [Figura 2.4] con el fin de mantener el CPU encendido, de lo contrario, si no existe un *wakelock* que bloquee el dispositivo entonces apaga la pantalla para conservar energía [9].

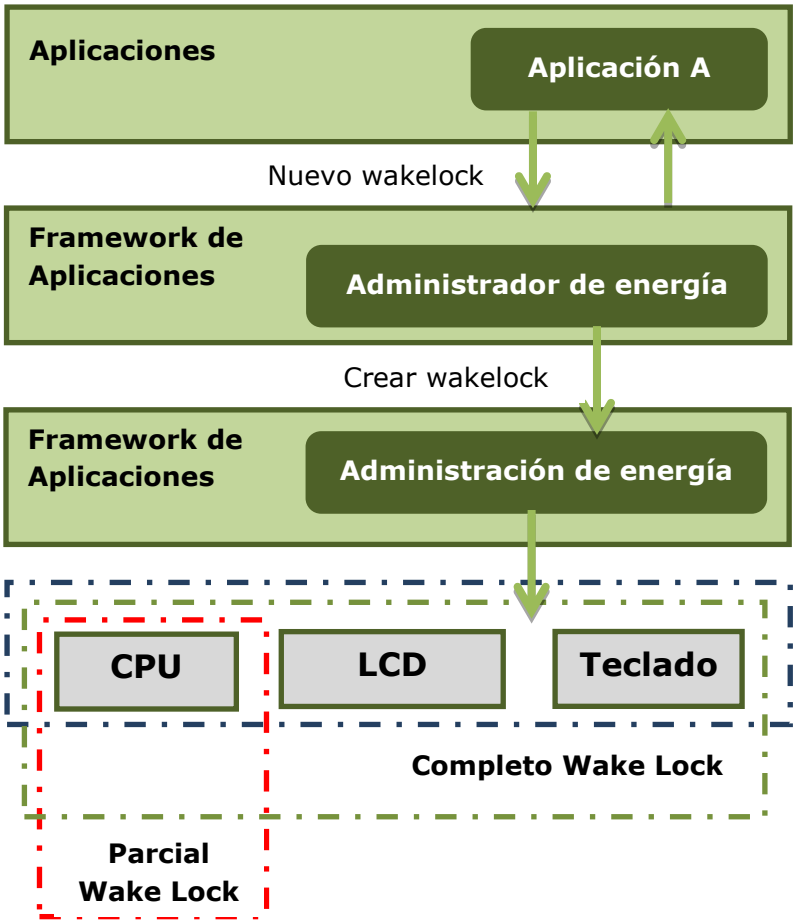


Figura 2.4. Arquitectura del administrador de energía de Android con wakelocks [9].

Wakelocks son un mecanismo del administrador de energía de Android, con él se puede controlar la pantalla, su brillo, el teclado y las luces de fondo de los botones. Un ejemplo de esto es cuando en una aplicación de juegos necesita que la pantalla del dispositivo permanezca encendida por un periodo determinado. El número de tipos de wakelocks podría no ser suficiente para el consumo de energía que tienen diversos dispositivos de entrada-salida, por

ejemplo, la antena WiFi es un dispositivo que consume una alta cantidad de energía, pero Android no apaga automáticamente el WiFi si el usuario no lo está utilizando.

2.2. Android SDK

Google mediante la herramienta Android SDK permite a los desarrolladores crear aplicaciones usando el lenguaje Java [Anexo C], para poder ejecutarlos también desarrollaron una máquina virtual llamada Dalvik que ejecuta archivos en el formato Dalvik Executable (.dex).

2.2.1. Máquina virtual Dalvik

Google optó por desarrollar su propia máquina virtual a pesar de que existen dos alternativas: la máquina virtual de java (JVM) y su versión móvil J2ME. Estas no se utilizaron debido a la velocidad de ejecución de las mismas ya que no están diseñadas para tomar ventaja de los teléfonos inteligentes de hoy en día, además el dispositivo móvil que alojara estos programas posee recursos limitados de hardware. Si lo comparamos con una PC se dan tres diferencias principales: el poder de la batería, refrigeración pasiva y el tamaño físico del dispositivo [10].

La dependencia de las baterías significa que la energía es escasa y por lo tanto debe ser conservada, la refrigeración pasiva limita la cantidad de potencia que se puede utilizar en cualquier momento dado para evitar causar que el dispositivo se caliente excesivamente, un dispositivo móvil debe ser ligero, por ello el peso, tamaño de la batería y otros componentes lo limita de igual forma, esto conduce a un sistema que fácilmente puede quedarse sin memoria o agotar la batería, si no se gestionan adecuadamente.

La seguridad es un punto de importancia debido a que los dispositivos móviles contienen una cantidad significativa de información sensible acerca del usuario: números de teléfono, registros de llamadas, ubicación física, etc. Por lo tanto, la gente espera un mayor nivel de seguridad para un teléfono inteligente que un teléfono tradicional en función de proteger estos datos. Dalvik ha sido diseñado para tomar ventaja del modelo de alta seguridad del núcleo de Linux, ya que cada instancia de la máquina virtual se ejecuta en su propio proceso.

En el diseño de Dalvik se optimizan algunos aspectos como el tamaño del archivo en bytecode, la velocidad del intérprete, uso de memoria, modificaciones en el diseño del colector de

basura y el inicializador de procesos llamado **Zygote**, el cual permite tener varias instancias de la máquina virtual Dalvik para compartir la memoria y así tener control de esta cuando varias aplicaciones corren concurrentemente, el propósito de Zygote es proporcionar un proceso precargado, sin incurrir en la demora de empezar la carga de instrucciones de memoria desde cero. Cuando el zygote crea un proceso, las clases principales son precargados en memoria. Como cada Zygote utiliza el comando *copy-on-write* para el espacio de memoria compartida, gran parte de la memoria para cada aplicación se reutiliza y por lo tanto supone una carga más ligera para los limitados recursos del dispositivo. Esto sólo funciona porque estas clases principales son en su mayoría de sólo lectura y rara vez se requiere más de una versión para todas las aplicaciones. Una vez que el zygote se ha habitado por una nueva aplicación, otro Zygote se crea a la esperara de que la siguiente aplicación sea lanzada.

Para llevar a cabo la recolección de basura se debe guardar un bit que represente cuando un objeto puede ser recogido, a este le llamamos "bit de marca". Estos bits se pueden almacenar ya sea junto al objeto en memoria o en conjunto en una tabla. Puesto que la memoria es tan limitada, es importante mantener tanta memoria compartida entre los

procesos como sea posible, esto significa que Dalvik debe evitar tener copias privadas generadas por bits de marca cambiantes para los objetos individuales en procesos separados que son en verdad, el mismo objeto. En su lugar, la máquina virtual utiliza una sola tabla en cada proceso que permite que los objetos permanezcan en la memoria compartida como una sola copia, a menos que sean modificados en cuyo punto se duplican. En comparación, el uso de una sola tabla para todos los procesos podría obligar a que los objetos sean retenidos más tiempo del necesario y más importante aumentar significativamente la complejidad del diseño.

2.2.2. Localización y WiFi

Para poder obtener las señales de localización se utilizó el GPS del dispositivo móvil. Estas señales se manipulan a través del SDK de Android, el cual ofrece diferentes formas de obtener la ubicación geográfica del usuario mediante tres proveedores que ofrecen este servicio [13]. El primero provee la ubicación mediante GPS, este proveedor determina la localización mediante satélites. Dependiendo de las condiciones, este proveedor puede tomar de 30 segundos a dos minutos [14] para devolver una posición. El segundo, es el

proveedor de ubicación mediante red celular, este proveedor de ubicación determina según la disponibilidad de las antenas de telefonía móvil y los puntos de acceso WiFi. Los resultados son obtenidos por medio de una búsqueda de red celular. El tercero, es un proveedor de ubicación especial, este proveedor se puede utilizar para recibir actualizaciones de localización de manera pasiva es decir, cuando otras aplicaciones o servicios soliciten la ubicación del usuario este proveedor devolverá ubicaciones generadas por otros proveedores, si el GPS no está activado se devuelve la ubicación basada en la red celular. Estos mecanismos tienen precisión, velocidad y consumo de recursos distintos los cuales se describen en la tabla II.

Tabla II: Tipos de Proveedores del servicio GPS y sus características

Proveedor	Características Tecnología	Datos
GPS autónomo	1. Usa chip GPS en el dispositivo 2. Línea de visión de los satélites 3. Necesita 7 satélites para fijar posición 4. Demora un tiempo mayor en fijar la posición 5. No funciona con edificios altos alrededor	Tiempo: 30 seg. a 2 min. Margen de Error: 6 m. Uso batería: Alto

Tabla II (continuación): Tipos de Proveedores del servicio GPS y sus características

Proveedor	Características Tecnología	Datos
Red Celular: (AGPS) GPS Asistido	1. Usa el chip de GPS del dispositivo y a su vez la ayuda de la red celular para proveer una fijación de posición inicial rápida 2. Bajo consumo de batería 3. Muy preciso 4. Depende de la portadora y el soporte (aun así si el teléfono soporta esta tecnología pero la red no, esta no funcionaría)	Tiempo: < 30 seg. Margen de Error: 60 m. Uso batería: Mediano-Bajo
Pasivo: Búsqueda de CellID , Búsqueda de WiFi MACID	1. No requiere batería extra 2. Tiene baja precisión, a veces tiene mayor precisión en lugares con muchos puntos de acceso WiFi y personas que comparten su ubicación con Google.	Tiempo: Depende de los anteriores. Margen de Error: 1615 m. Uso batería: Bajo

2.3. Android NDK

Las aplicaciones para el sistema operativo Android son típicamente desarrolladas en lenguaje de programación Java. Sin embargo a veces es necesario superar las limitaciones que este lenguaje posee, como por ejemplo el manejo de memoria o el rendimiento de procesos ejecutados en una máquina virtual. Esto se lo puede lograr mediante la programación directamente en la interfaz nativa de Android. Para esto Android proporciona el NDK que es un conjunto de herramientas que permiten a los desarrolladores implementar

partes de sus programas en lenguajes C y C++. Esto permite invocar llamadas al sistema para comunicarse directamente con el sistema operativo, para tener acceso directo a los sensores del teléfono, emplear mecanismos POSIX de comunicación entre procesos y utilizar técnicas de programación concurrente como la sincronización entre procesos. A continuación se explican las técnicas utilizadas en nuestro proyecto.

Un **socket** es un punto de comunicación por el cual un proceso puede emitir o recibir información hacia procesos ejecutándose en diferentes computadoras conectadas a una misma red. En los sistemas operativos basados en Unix un *socket* se identifica por un descriptor de archivo, del mismo tipo que el utilizado para referenciar a los archivos del sistema operativo anfitrión. Sobre el cual se puede leer o escribir, permitiéndose una comunicación bidireccional, característica propia de los *sockets* y que los diferencia de los pipes los cuales son canales de comunicación unidireccional entre procesos de un mismo computador.

La comunicación entre procesos a través de *sockets* se basa comúnmente en el modelo cliente-servidor [Figura 2.5], en donde un proceso actúa como servidor creando el *socket*, a su vez la información del mismo la conoce otro proceso que actúa como cliente, el cual podrá comunicarse con el servidor a través de la

conexión con dicho *socket*. Los *sockets* son capaces de utilizar el protocolo de streams TCP (Transfer Control Protocol) y el de datagramas UDP (UserDatagramProtocol) [15].

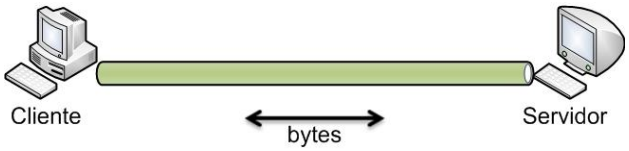


Figura 2.5. Comunicación cliente servidor.

Los **hilos** llamados en inglés “threads”, permiten a un programa ejecutar varias tareas de manera concurrentemente. El *Kernel* del sistema operativo interrumpe cada hilo de momento en momento para así dar oportunidad al resto de hilos de ejecutar parte de sus instrucciones hasta terminar [16]. Si se tiene más de un hilo en ejecución, estos comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos y variables globales. El hecho de que los hilos de un mismo proceso compartan los recursos hace que cualquiera de estos hilos pueda modificarlos. Cuando un hilo modifica un dato en la memoria, los otros hilos acceden a ese dato modificado inmediatamente [17].

Los **semáforos** son un mecanismo que permite resolver problemas de sección crítica entre procesos, que se dan comúnmente al emplear programación concurrente. Por ejemplo, las condiciones de

carrera, que ocurren cuando varios procesos acceden y manipulan los mismos datos concurrentemente, y el resultado final depende del orden particular en que los accesos se llevaron a cabo.

El semáforo es una variable entera que es modificada por dos operaciones: *wait()* y *signal()*, las cuales disminuyen y aumentan el semáforo respectivamente. El sistema operativo garantiza que dos procesos no puedan ejecutar *wait* y *signal* en el mismo semáforo, al mismo tiempo. Por lo tanto, implementar un semáforo se torna en el problema de la sección crítica, en donde el código de *wait* y *signal* se colocan dentro de esta sección. La sección crítica es la sección del programa en donde varios procesos acceden a los datos compartidos [Figura 2.6]

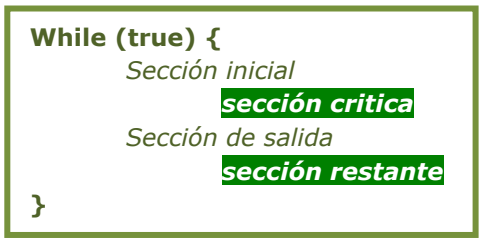


Figura 2.6. Estructura de un proceso con sección critica.

Tanto los *sockets*, hilos y semáforos tienen soporte en el NDK de Android. Para comprobar esto realizamos al inicio del proyecto pruebas con porciones de código recibidas en las clases teóricas, las cuales fueron demostradas en el capítulo 4 de resultados.

CAPITULO 3

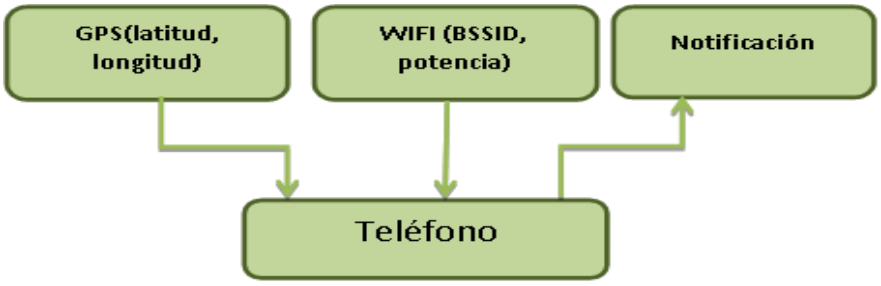
3. IMPLEMENTACIÓN

Se decidió como proyecto una aplicación móvil utilizando tanto el SDK como el NDK. En esta se implementaron los conceptos sobre programación concurrente.

3.1. Diseño de la solución

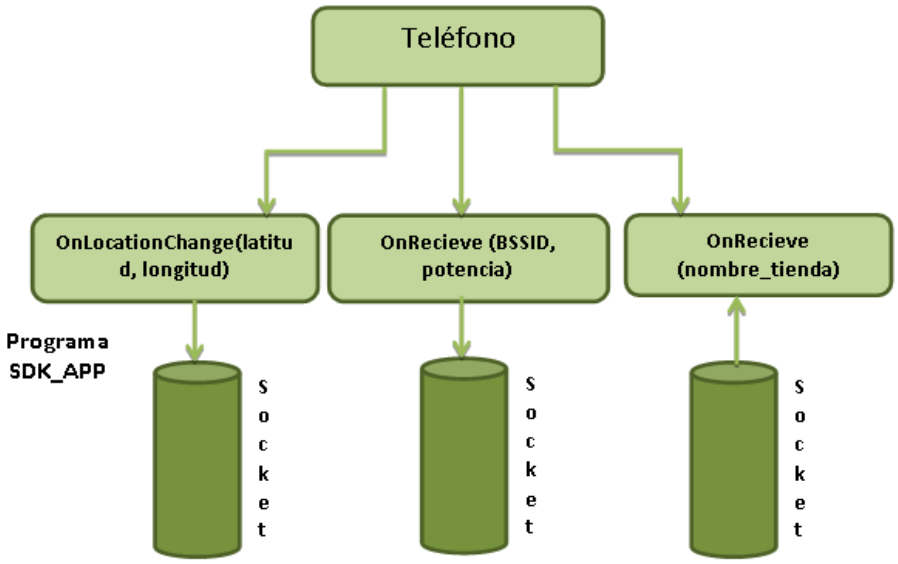
Basándonos en los conceptos previamente descritos diseñamos nuestro proyecto siguiendo el esquema de la Figura E.1. En esta se nombra al programa escrito en Java como programa SDK_APP y al programa escrito en C como programa NDK_APP.

En nivel uno de la Figura E.1 se encuentra la aplicación móvil en donde el usuario recibe señales GPS y señales WiFi, ambas son recibidas por el programa SDK_APP para su respectivo proceso. Además este nivel incluye una notificación al usuario de cuando se llega a un lugar o tienda en donde vendan el artículo deseado.



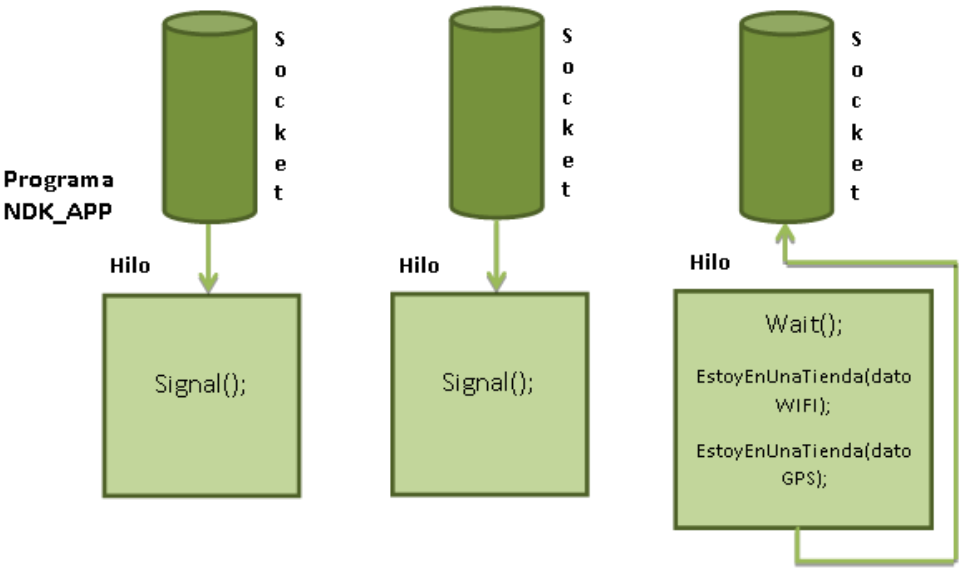
Nivel 1 Figura E.1.

En el programa SDK_APP usamos dos métodos encargados de recibir los datos, como podemos ver en la nivel dos la Figura E.1. son la latitud y longitud para el GPS y el BSSID y el nivel de potencia para el WiFi. Además implementamos otro método encargado de recibir el nombre de local enviado desde el programa NDK_APP. A su vez contamos con tres *sockets* los cuales sirven de comunicación con el programa NDK_APP cuando se requiera enviar o recibir algún dato.



Nivel 2 Figura E.1.

En el programa NDK_APP tenemos así mismo tres *sockets* los cuales se conectan con los *sockets* creados con el programa SDK_APP tal como nuestra el nivel tres de la Figura E.1. La información de Java es manejada por tres hilos, para la sincronización usamos semáforos, estos son utilizados para que uno de los hilos espere hasta que exista algún dato nuevo por procesar, cuando exista algún dato nuevo este hilo lo procesa y dependiendo o no del resultado envía el nombre del local encontrado por medio del socket.



Nivel 3 Figura E.1.

3.2. Acceso a sensores

Una de las complicaciones del proyecto fue el acceso a los sensores del teléfono móvil por medio del NDK, debido a que existe una lista específica de los sensores a los cuales podemos acceder de manera abierta, la misma que encontramos en la librería sensor.h de Android; sin embargo en la lista no constaban los sensores que requerimos para el proyecto como son el GPS y WiFi. Google argumentó que la única manera de acceder a estos sensores es mediante SDK (Java) [Anexo E], por ello se buscaron otras opciones, por ejemplo una nueva herramienta de Google, llamado Android PDK, que ayuda a las compañías dedicadas a la creación de hardware a tener una idea de que cambios hubieren al momento que Android realice un lanzamiento de una nueva versión y poder tener una actualización

propia en menor tiempo. Esta opción permite crear una librería según las necesidades del desarrollador sobre el sensor que se necesita.

Pero esto no resolvía la problemática, ya que era solo para fabricantes de dispositivos lo cual no era parte de nuestros objetivos, la siguiente opción a analizar fue la implementación de *sockets* el cual nos permite comunicar procesos creados a partir de los frameworks SDK y NDK, es decir, los datos se los obtendrá en el programa codificado en Java (opción única dada por Google) y el procesamiento de datos se lo haría en el programa escrito en lenguaje C pudiendo implementar las ventajas de la programación concurrente, además de que programando directamente en la interfaz nativa de Android logramos que nuestra aplicación tenga un desempeño mayor.

3.3. Procesamiento de datos en Programa Java

Como antes mencionamos los datos que necesitamos los obtenemos por medio del programa SDK_APP ya que no existe soporte en NDK para acceder nativamente a los datos. La forma en que se obtiene el dato del GPS es a través de dos clases de java la primera es *LocationManager* que es el que provee el acceso al servicio de localización de Android, la segunda es *LocationListener* que es utilizada para recibir notificaciones del *LocationManager* cuando ha

ocurrido un cambio de locación [18], estas son recibidas a su vez por un método asincrónico llamado *OnLocationChange* el cual recibe como parámetros un valor de latitud y uno de longitud correspondientes a la posición actual del teléfono. Cuando un cambio de locación es recibido dentro de este método, los nuevos datos de latitud y longitud son enviados por el socket previamente creado el cual actúa como puente para la comunicación con el programa NDK_APP.

Para obtener los datos del WiFi, hacemos uso de dos clases: La primera *WiFiManager* la cual sirve para manejar todos los aspectos de la conectividad WiFi como por ejemplo la lista de las redes configuradas o la red actualmente activa y la segunda *BroadcastReceiver* la cual tiene un método llamado *onReceive* que es asincrónico y se ejecuta cada vez que detecta una señal nueva de WiFi. Cuando *onReceive* se ejecuta, primero se obtiene una lista, a través del *WiFiManager*, con todas las redes WiFi que hayan sido detectadas luego son enviadas una a una a través del socket previamente creado, la información de la red que se envía es el BSSID y la potencia.

Para poder notificar al usuario que ha llegado a alguna tienda donde pueda adquirir el artículo deseado, primero el nombre de la tienda es enviado desde el programa NDK_APP, después de haber realizado la debida comparación, y este es recibido por medio del socket en el

programa Java. Cuando el dato llega se lo contrasta con una lista de nombres de tiendas soportados por el programa de esa manera se determina que artículos venden en esta tienda, si alguno de estos coincide con algún artículo que se encuentra en la lista previamente creada por el usuario, se le avisa por medio de una notificación propia del sistema operativo Android que se encuentra cerca de la tienda.

3.4. Comunicación entre procesos

En nuestro caso usamos *sockets* en el mismo dispositivo, es decir, con la misma dirección ip (127.0.0.1) pero diferentes puertos (2008, 2009 y 2010) para intercambiar datos entre procesos del programa SDK_APP y el programa NDK_APP. Para dicha tarea usamos un modelo cliente-servidor, se puso como primer escenario el programa SDK_APP que funciona como servidor enviando datos de GPS (latitud, longitud) y WiFi (dirección MAC, Potencia en decibeles) cada uno por *sockets* independientes, además tenemos el programa NDK_APP que funciona como cliente, recibiendo los datos para posteriormente procesarlos. Como segundo escenario, ahora el programa NDK_APP es el servidor que después de haber procesado los datos envía por un tercer socket el nombre del local donde se encuentra el usuario al cliente, que en este caso es el programa SDK_APP [Figura 3.1]. En el siguiente subcapítulo se explica con mayor detalle los procesos involucrados.

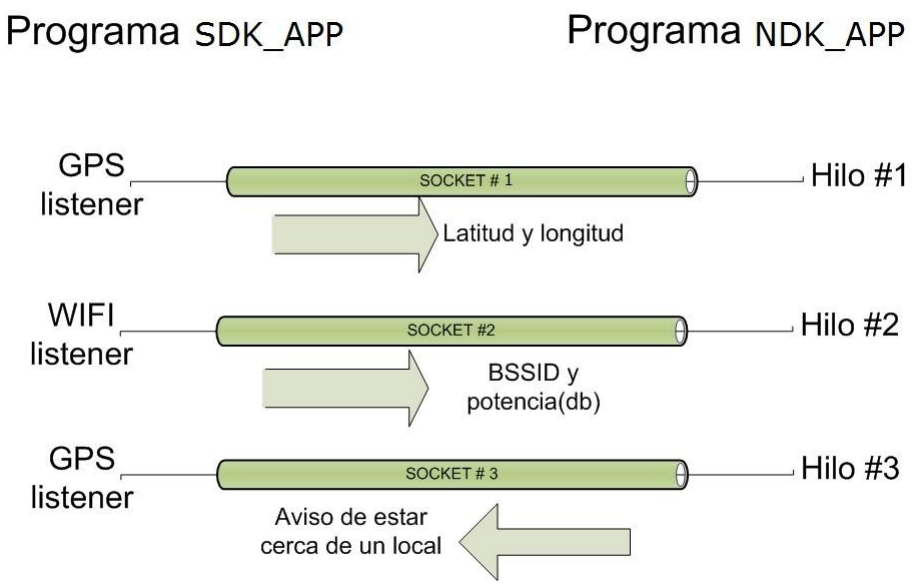


Figura 3.1. Funcionamiento de Sockets entre Programas en Java y C

3.5. Procesamiento de datos en programa C

Para el almacenamiento de datos se utilizó un archivo de texto que contenía la información de los principales locales del centro comercial, estos eran llevados a un arreglo de estructuras de datos que facilitaba su acceso, en la estructura se guardaba el nombre del local comercial, su posición geográfica (latitud y longitud) y las direcciones MACs con sus respectivas potencias pertenecientes a los enrutadores inalámbricos más cercanos al local, para que de esta forma se pueda tener un acceso más rápido a estos datos, con lo cual tendríamos algo muy similar a una base de datos.

Entonces cada vez que el programa SDK_APP enviaba algún dato de GPS o WiFi a través de los *sockets* estos eran comparados con

nuestro arreglo de estructuras. Si alguno de los dos datos (GPS o WiFi) coincidía con la información de alguno de los locales guardados previamente, se enviaba el nombre de dicho local por el socket hacia el programa SDK_APP.

3.6. Hilos

Para la implementación de hilos en el programa NDK_APP se usó la librería GNU *Pthread*, con la cual se realizaron pruebas iniciales para verificar el soporte en Android NDK. La aplicación consta con tres hilos, cada uno aloja un socket para la comunicación con el programa SDK_APP [Figura 3.2].

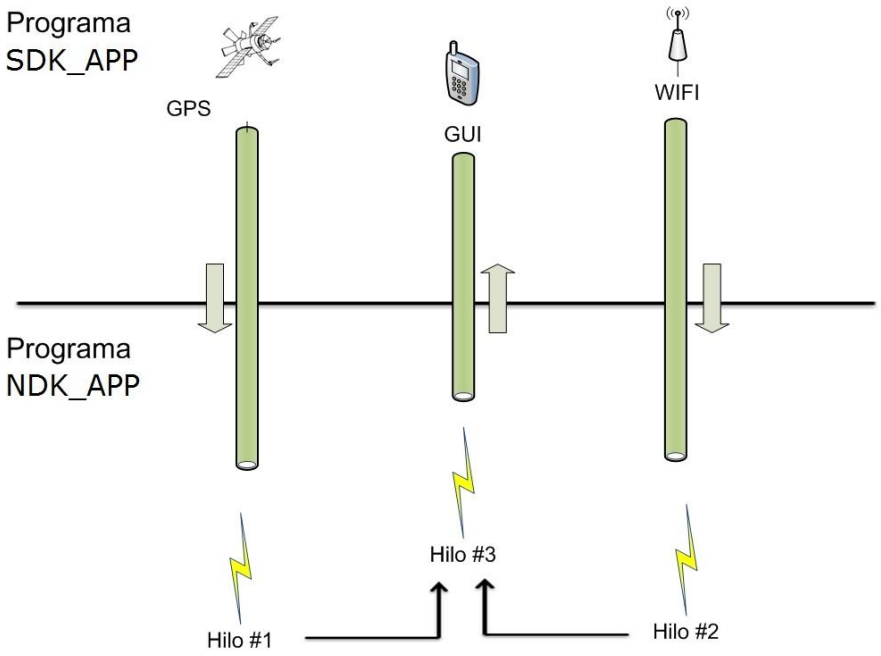


Figura 3.2. Funcionamiento de los hilos creados.

El primer hilo se encarga de alojar y mantener el socket por donde viaja la información del WiFi proveniente del programa SDK_APP, siendo esta la dirección MAC y el nivel de potencia de las redes más cercanas al local. El segundo hilo se encarga de igual forma alojar y mantener un socket pero este recibe información del GPS, es decir la latitud y longitud del teléfono.

Ambos hilos utilizan una sentencia propia de los *sockets* llamada *read* la cual hace que el hilo que contiene al *socket* se suspenda hasta que reciba algún dato, esta sentencia se realiza dentro de un lazo para que pueda recibir futuros datos. El tercer hilo al igual que los anteriores aloja y mantiene un *socket* pero como se mencionó antes este envía al programa SDK_APP el nombre del local comercial desde el programa NDK_APP.

Los tres hilos fueron creados en un mismo proceso por ende comparten la memoria y los archivos de este, facilitando la comunicación entre ellos. En este caso los tres hilos comparten dos variables, una que sirve para guardar el dato de GPS y la otra para guardar el dato de WiFi, ambos datos provenientes del programa SDK_APP. Estas variables son consultadas constantemente por el tercer hilo cuando se requiera hacer el proceso de comparación. Si bien, no existe una sección critica en nuestro programa NDK_APP, en donde más de un hilo modifique el valor de una variable, existe

una sección en donde se lee el contenido de las variables, por eso lo que hacemos es sincronizar las consultas que se le realicen a estas, es decir que se hagan en correcto orden y cuando sea necesario, para así evitar inconsistencia en los resultados. El detalle de cómo se realizó esta sincronización se explica en el siguiente subcapítulo.

3.7. Sincronización de Hilos

Se empleó semáforos para coordinar el trabajo de los tres hilos antes mencionados. Los dos hilos que reciben información desde el programa SDK_APP escriben en dos variables globales diferentes, como ya se había mencionado existe además otro hilo encargado de leer estas variables para luego compararlas con las estructuras.

El problema radicaba en que esta comparación no se debe realizar a menos que existan datos que comparar o si estos han sido actualizados previamente. A continuación explicaremos el escenario en que los semáforos logran sincronizar el acceso a las variables globales antes mencionadas [Figura 3.3]:

Encontramos a los hilos que manejan los datos del GPS y WiFi se encuentran suspendidos por medio del comando *read* utilizado para la comunicación en los *sockets*, a la espera de algún dato nuevo enviado desde el programa SDK_APP. A su vez el hilo que envía

datos al programa SDK_APP se encuentra suspendido por medio del comando *wait*. Cuando se recibe desde el programa SDK_APP algún dato de GPS o WiFi alguno de los dos hilos encargados leerán este dato, dejaran de estar suspendidos y ejecutaran el comando *signal* lo que provocara que el hilo que se encontraba suspendido por medio del *wait* salga de ese estado y realice la comparación debida de los datos recibidos con nuestras estructuras de datos.

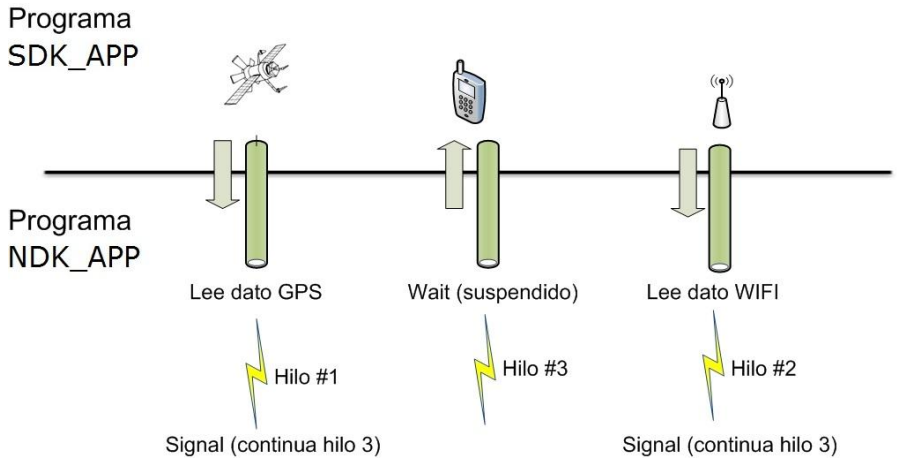


Figura 3.3. Funcionamiento Semáforos

CAPITULO 4

4. RESULTADOS

Se realizaron cuatro pruebas con diferentes objetivos cada una, las dos últimas pruebas fueran realizadas en un ambiente de campo, las cuales se detallan a continuación.

4.1. Prueba 1: Soporte de la librería de hilos Pthread en Android NDK

La primera prueba que se realizo fue con el objetivo de verificar si existía soporte en Android NDK para la librería de hilos *Pthread*.

4.1.1. Descripción de la prueba

Dado que Android es POSIX compatible deberíamos poder ejecutar en Android NDK el código de los ejemplos que hemos visto clase, los cuales consistían en la creación de hilos/procesos que se ejecutaban concurrentemente para realizar una operación. El código que nosotros utilizamos consistía en la creación de 14 hilos que accedían a una variable, aumentaban su valor y al final esta se imprimía por pantalla, tal como muestra la figura 4.1 y 4.2.

```
#include <pthread.h>
#include <stdio.h>
int sum;
void *runner(void *param);
int main(int argc, char *argv[]){
        pid_t cpid, w;
        int status,i;
        pthread_t tid;
        pthread_t tida[15];
        pthread_attr_t attr;
        pthread_attr_init(&attr);
        for (i=1;i<15;i++)

        pthread_create(&tida[i],&attr,runner,NULL);
        for (i=1;i<15;i++)
                pthread_join(tida[i],NULL);
        printf("sum = %d\n",sum);
return(0);  }
void *runner(void *param) {
        int i, upper = 5;
        if (upper > 0) {
                for (i = 1; i <= upper; i++)
                        sum += i;  }
        pthread_exit(0); }
```

Figura 4.1. Código C en Linux.

```
#include <jni.h>
#include <android/log.h>
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#define DEBUG_TAG "NDK_GEOMEMO_ACTIVITY"
int sum=0; /*
int NUM_THREADS=15;
void Java_com_tigsesolis_geomemo_GeoMemoActivity_helloLog(JNIEnv *
env, jobject this, jstring logThis)   {
            pid_t cpid, w;
            int status,i;
            jboolean isCopy;
            const char * szLogThis = (*env)->GetStringUTFChars(env, logThis,
&isCopy);
            __android_log_print(ANDROID_LOG_DEBUG, DEBUG_TAG,
"NDK:Imprimiendo desde C: [%s]", szLogThis);
            (*env)->ReleaseStringUTFChars(env, logThis, szLogThis);
            pthread_t tid;
            pthread_t tida[NUM_THREADS];
            pthread_attr_t attr
            pthread_attr_init(&attr);
            for (i=1;i<NUM_THREADS;i++)
                        pthread_create(&tida[i],&attr,runner,NULL);
            for (i=1;i<NUM_THREADS;i++)
                        pthread_join(tida[i],NULL);
            __android_log_print(ANDROID_LOG_DEBUG, DEBUG_TAG,
"sum = %d\n",sum);
}
void *runner()  {
int i, upper = 5;
            if (upper > 0) {
                        for (i = 1; i <= upper; i++)
                                    sum += i;
      }
      sleep(30);
            pthread_exit(0);
}
```

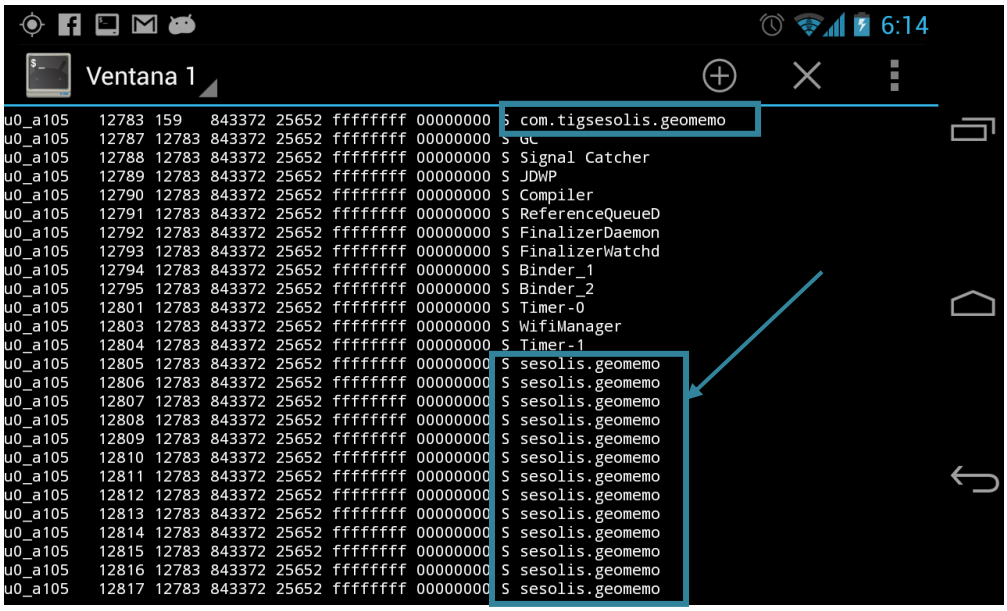
Figura 4.2. Código C usando Android NDK

Como podemos observar entre el código del lenguaje C y android NDK existen pequeñas diferencias. Como son la función para impresión por pantalla y el prototipo de las funciones. Esto nos da la ventaja de reutilizar código ya creado en C en Android NDK.

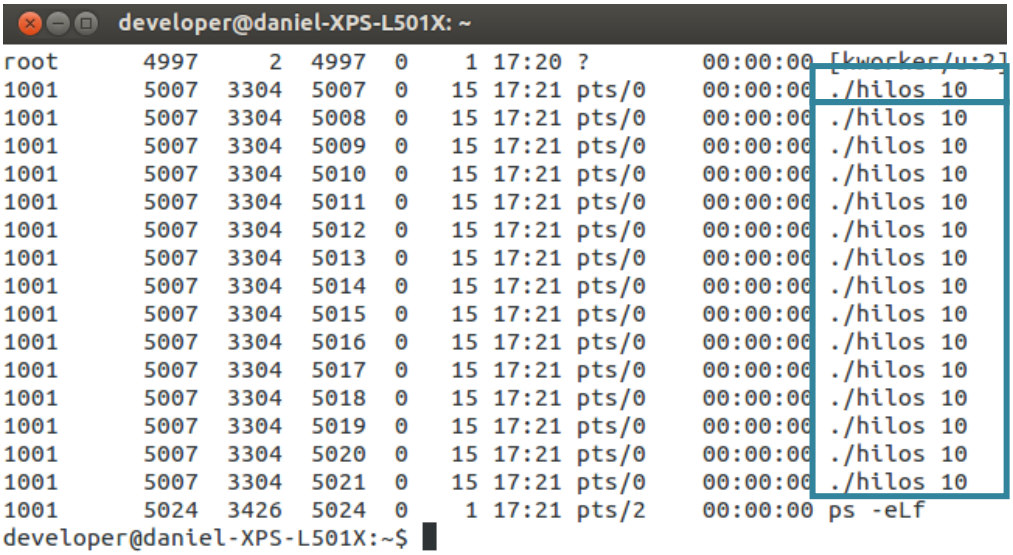
4.1.2. Resultados Obtenidos

A continuación se pueden observar dos capturas de pantalla [Figura 4.3], en la primera apreciamos el resultado de ejecutar el programa en el sistema operativo Linux y en la segunda se muestra el resultado de ejecutar el programa en el sistema operativo Android.

Para poder visualizar el resultado del programa en Android se usó una aplicación en el teléfono móvil llamada “Terminal” que simula una consola que nos permite ejecutar comandos de Linux.



a.



b.

Figura 4.3. Comparación de resultados al ejecutar Hilos en NDK (a) y en la consola de Linux (b).

Tanto en Android NDK como en Linux los hilos respondieron de la misma manera ya que en ambos programas como muestra la figura 4.3 se pueden ver la misma cantidad de hilos en ejecución. En NDK el nombre con que se imprime el programa es *sesolis.geomemo* y en Linux es *./hilos.* Para mostrar los hilos ejecutándose en Android NDK insertamos una porción de código que me permitía ejecutar comandos de Linux, el comando ejecutado fue: *ps*. En Linux en cambio para mostrar los hilos en ejecución utilizamos el comando: *ps –eLf.* Para calcular los tiempos en Android NDK lo que hicimos fue visualizar el tiempo de ejecución de cada hilo en la ventana de Log que provee el entorno de desarrollo Eclipse. En cambio

para calcular los tiempos en Linux utilizamos el comando: *time*. Los tiempos promedio de ejecución obtenidos en la prueba fueron de 0,004s con una desviación estándar de 0,0004714 en la consola de Linux y de 0,005s con una desviación estándar de 0,00056765 en el programa de Android NDK, cabe mencionar que los tiempos de ejecución en ambos casos van a depender del hardware del dispositivo o PC.

En Android NDK la compilación del código se realizó con un compilador propio de Google llamado *ndk-build*, en cambio, la compilación en Linux se realizó mediante el compilador de lenguaje en C en Linux llamado *gcc*.

4.2. Prueba 2: Comunicación entre procesos usando sockets

En esta prueba el objetivo era evaluar el establecimiento de conexión y el envió de datos a través de *sockets* entre un proceso del programa desarrollado en SDK (Java) y un proceso del programa desarrollado con las herramientas del NDK (C), llamados en el capítulo 3 SDK_APP y NDK_APP respectivamente.

4.2.1. Descripción de la prueba

Esta prueba consistió en recopilar diferentes latitudes y longitudes obtenidas del GPS del dispositivo móvil así como diferentes direcciones MACs y potencias de redes WiFi de los locales comerciales obtenidas desde la tarjeta inalámbrica del dispositivo móvil. Estos datos llegaban al proceso que corresponde al programa SDK_APP al que llamaremos servidor y eran enviados a través de los *sockets* al proceso que corresponde al programa NDK_APP al que llamaremos cliente. El cliente al ejecutarse trata de conectarse con el servidor pero si no puede realizar esta tarea termina, el servidor abre un *socket* y espera una conexión del cliente. Los *sockets* creados para la comunicación son tipo TCP, utilizamos la dirección de *localhost* (127.0.0.1) ya que la comunicación es dentro del mismo dispositivo tal como se menciona en el capítulo tres de implementación.

4.2.2. Resultados Obtenidos

Al inicio se realizaron 10 pruebas, de estas solo 6 veces se creó el *socket* sin problemas, pero el resto de veces el cliente se creaba antes que el servidor por ende no lograba

conectarse correctamente [Figura 4.4]. Cuando esto sucedía el cliente terminaba de ejecutarse y el resto del programa no funcionaba correctamente.

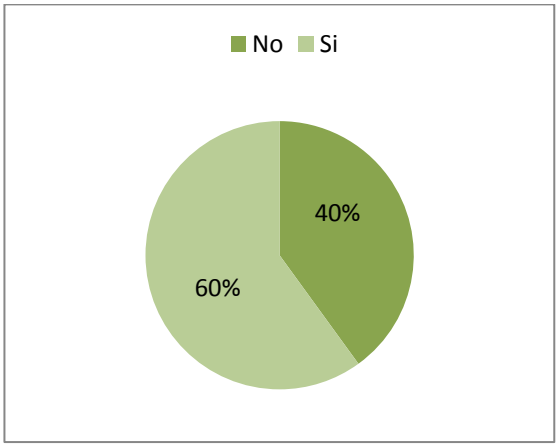


Figura 4.4. Diagrama de resultados de prueba de conexión de sockets.

La solución fue modificar el código de la aplicación, es decir, dejar que lo primero que se ejecute sean las funciones para mostrar la interfaz gráfica y al último la llamada al cliente, de esta forma el servidor siempre estaba listo para recibir conexiones del cliente, teniendo en cuenta que la llamada al programa NDK_APP se realiza dentro del programa SDK_APP tal como se explica en el capítulo tres de implementación.

Una vez que se estableció la conexión, tomamos el tiempo en 20 muestras desde que llegan los datos de GPS o WiFi al servidor hasta que estos era recibidos en el cliente dando

como resultados [Figura 4.5] que los datos obtenidos desde la tarjeta inalámbrica de WiFi tardaban en enviarse un promedio de 3,75 milisegundos con una desviación estándar de 1,08 y en cambio los datos obtenidos del GPS del dispositivo móvil tardaban 3,38 milisegundos con una desviación estándar de 0,83. Esto podría deberse a la cantidad de datos en el contenido de la trama. Un ejemplo de contenido de la trama GPS sería el siguiente: -79.892891,-2.155478 correspondiente a la latitud y longitud y del contenido de la trama WiFi dc:9f:db:0a:9f:46,-59 correspondiente a la dirección MAC y nivel de potencia. En ambos casos separábamos los datos por comas.

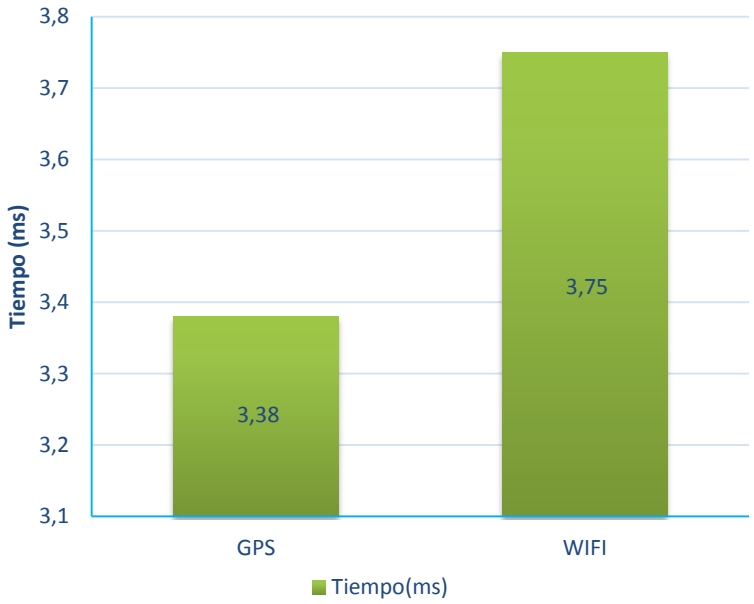


Figura 4.5. Tiempo promedio de enviar datos vía sockets entre servidor (SDK) y cliente (NDK).

4.3. Prueba 3: Obtención de información para base de datos

La prueba que se realizo fue en el centro comercial elegido (Mall del Sol) donde recorrimos los siguientes locales: Fybeca, Megamaxi, Etafashion, Musicalísimo y Juguetón registrando los datos que necesitamos para llenar nuestro archivo que servirá como base de datos de la aplicación.

4.3.1. Descripción de la prueba

Los datos recolectados fueron el nombre de la tienda, sus coordenadas geográficas y las cuatro redes con mayor nivel de potencia cercanas a la tienda, de estas redes registramos su dirección Mac y nivel de potencia.

El objetivo de esta prueba también es comparar y obtener un criterio de búsqueda tanto para los datos WiFi como para los datos del GPS que nos ayuden a definir de mejor manera la posición de una persona con respecto a una tienda.

Para obtener las 4 redes con mayor de nivel de potencia nos situábamos sobre el punto C, dado que las redes que se detectan en el centro son las que más nos favorecen [Figura 4.6].

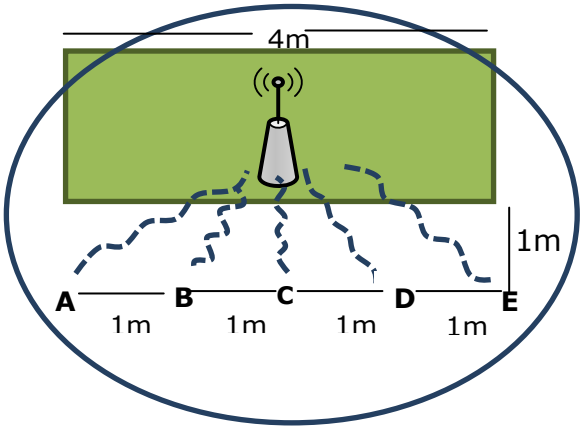


Figura 4.6. Ejemplo de un local con su propio enrutador.

En el punto C utilizamos una aplicación descargada del Google Play Store llamada *WiFi Analayzer* que nos mostraba la dirección Mac y nivel de potencia de estas redes. Estas cuatro redes seleccionadas solo podían ser puntos de acceso debido a que las señales receptadas también correspondían a otros tipos de dispositivos, tales como televisores, impresoras, etc. Como muestra la tabla V. Luego definimos cinco puntos diferentes próximos a la tienda, estos puntos están distribuidos a una distancia proporcional al ancho de la tienda.

Tabla III. Principales redes obtenidas en el punto C de una tienda.

Punto C		
Redes	**Tipo**	**Potencia(db)**
CNT-MALL DEL SOL	AP	-95
DIRECT-HC-BRAVIA	TV	-51
FUROIANI-MALL-SOL	AP	-73
HPE910.5B	IMPRESORA	-54
JUAN_VALDEZ	AP	-96
LG-WiFi	TV	-68
MSOL_WIRELESS	AP	-85
MUSI_MALL SOL	AP	-74
PACIFICARD	AP	-52
MTP_MALL SOL	AP	-61

Por cada red, y por cada punto anotábamos 20 mediciones del nivel de potencia obtenido con la aplicación antes mencionada, el número de mediciones es estadísticamente valido. Cuando teníamos los datos en los cinco puntos, sacábamos una media y una desviación estándar. Estos dos datos eran usados para obtener un rango ya que asumiendo que nuestro conjunto de datos se distribuye de manera “normal” podemos inferir que el 95% de las muestras tienen un valor que se encuentra a más-menos dos veces la desviación estándar.

La red inalámbrica de la tienda no siempre era tomada en cuenta debido a que su nivel de potencia no estaba entre los cuatro más altos o simplemente la tienda no tenía un punto

acceso propio, por ende los que si eran tomados en cuenta tenían una ubicación distinta a la tienda.

4.3.2. Resultados Obtenidos

En el siguiente gráfico se muestran los niveles de potencia obtenidos en los cinco puntos de una de las tiendas y las barras de error que representan a la desviación estándar de cada red [Figura 4.7].

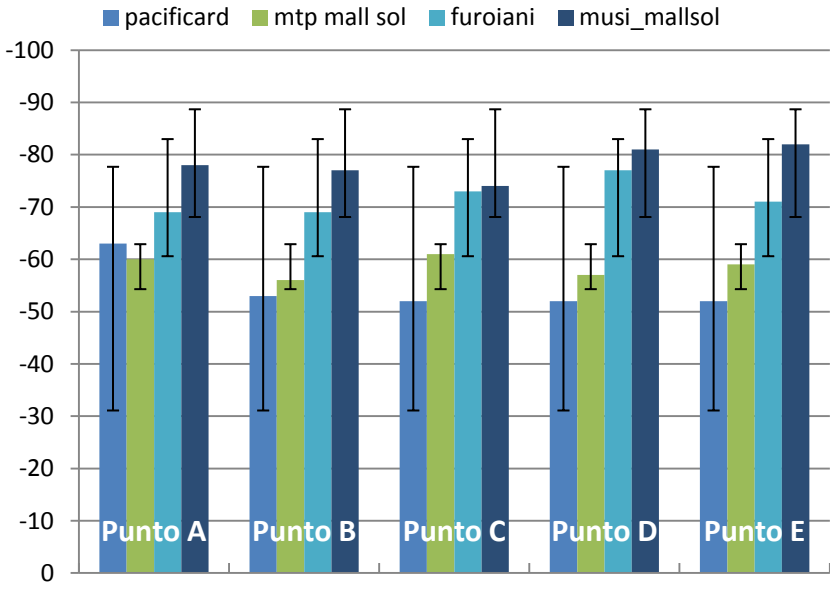


Figura 4.7. Niveles de potencia obtenidos y barras de error de una de las tiendas.

Como se puede observar en la figura 4.5 en algunas redes existe mayor valor de desviación estándar debido a que la potencia de las señales variaba mucho en el tiempo. Por esta

misma razón los niveles de potencia no aumentan ni disminuyen secuencialmente tal como muestra la tabla VI.

Tabla IV. Nivel de potencia señal vs tiempo (seg).

	Pacificard	mtp mall sol	furoiani	musi_mallsol
0	-52	-61	-73	-74
1	-50	-58	-70	-74
2	-54	-65	-68	-77
3	-65	-60	-75	-78
4	-52	-63	-74	-72
5	-60	-60	-70	-71

Entre los motivos para la variación en los niveles de potencia se encuentran: el ambiente, la cantidad de personas que transitan en los pasillos, la decoración de madera en los exteriores de los locales y las redes que trabajan en los mismos canales WiFi tal como se aprecia en la Figura 4.8, lo que supone un rendimiento más lento y una mayor pérdida de paquetes de datos.

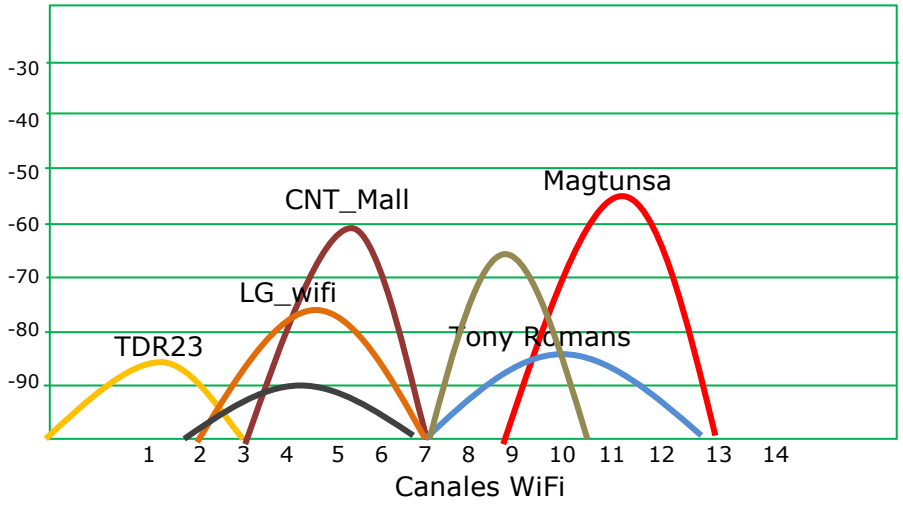


Figura 4.8. Potencia señal vs canales de WiFi

Pudimos también notar que en el caso de que dos o más tiendas estén cerca [Figura 4.9], el rango del nivel de potencia de alguna de sus redes tendría coincidencias lo cual puede causar que no siempre se detecte correctamente a una tienda.

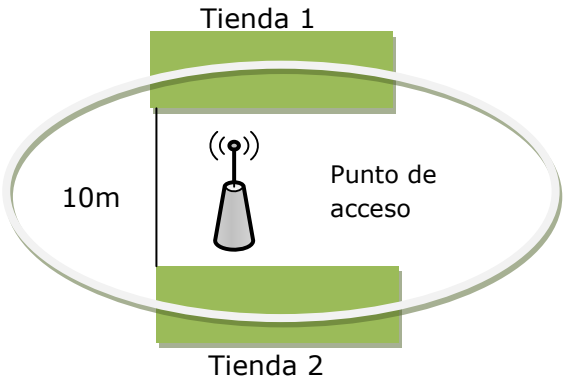
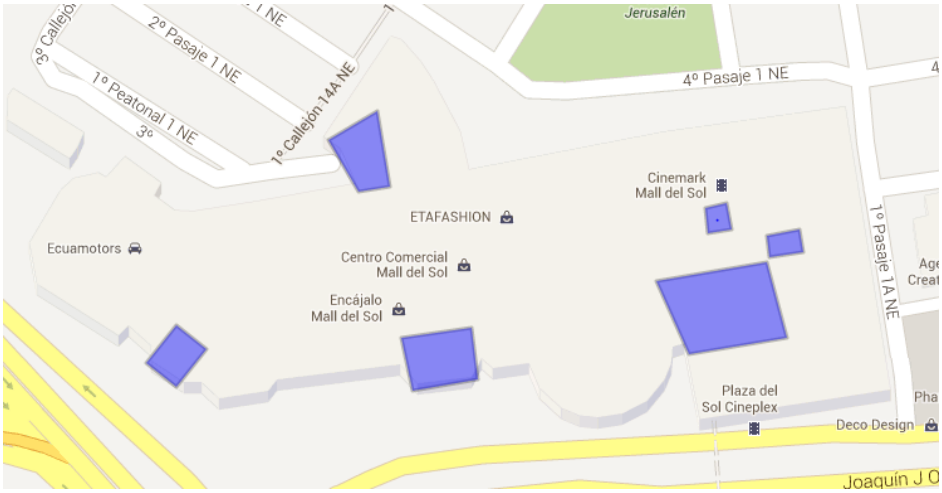


Figura 4.9. Tiendas con rangos de nivel de potencia parecidos

Los datos de latitud y longitud fueron obtenidos utilizando un mapa del centro comercial Mall del Sol [19] y un mapa obtenido de la herramienta web Google Maps. En la página de Google Maps localizamos al centro comercial y creamos áreas correspondientes a 5 tiendas. Las coordenadas de estas áreas tienen un valor aproximado. La idea fue comparar este mapa con el mapa que teníamos de los locales comerciales [Figura 4.10] y tratar que la posición de las áreas que creábamos sean las mismas.

a.



b.

Figura 4.10. Mapa de locales de Mall del Sol (a) y Mapa obtenido de Google Maps (b).

Una vez que teníamos este mapa creado en Google Maps, lo exportamos a un formato kml [Anexo F], que es muy parecido al formato xml, en donde se muestra la información en forma de etiquetas lo cual es fácil de interpretar. De este archivo se pudo extraer la información de latitud y longitud de los locales comerciales.

4.4. Prueba 4: Evaluación del proyecto

Esta última prueba de igual forma fue realizada en un centro comercial, ya con la aplicación finalizada.

4.4.1. Descripción de la prueba

Se guardaron 10 entradas, eligiendo diferentes categorías, sub categorías y un campo llamado detalle el cual indicaba que se debía comprar algún artículo. Después caminamos por distintos pasillos del centro comercial para verificar que la aplicación mostrara en pantalla el nombre de la tienda más cercana a nosotros. Luego comprobamos que la aplicación enviaba una notificación al usuario al pasar por alguna tienda donde vendían el artículo guardado en el GeoMemo. Estas tiendas corresponden a las tiendas analizadas en la prueba anterior.

El objetivo de esta prueba es calcular la probabilidad con que nuestra aplicación detecta la posición de una persona con respecto a una tienda correcta, es decir la que tiene el artículo que se desea comprar.

4.4.2. Resultados Obtenidos

Luego de pasar 10 veces por las 5 tiendas elegidas del centro comercial la probabilidad con la que nuestra aplicación GeoMemo acierta se resume en el siguiente gráfico [Figura 4.11].

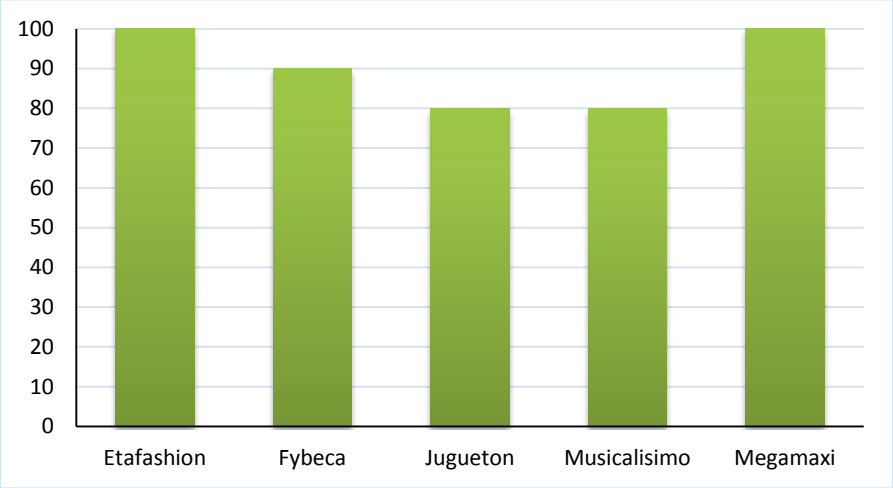
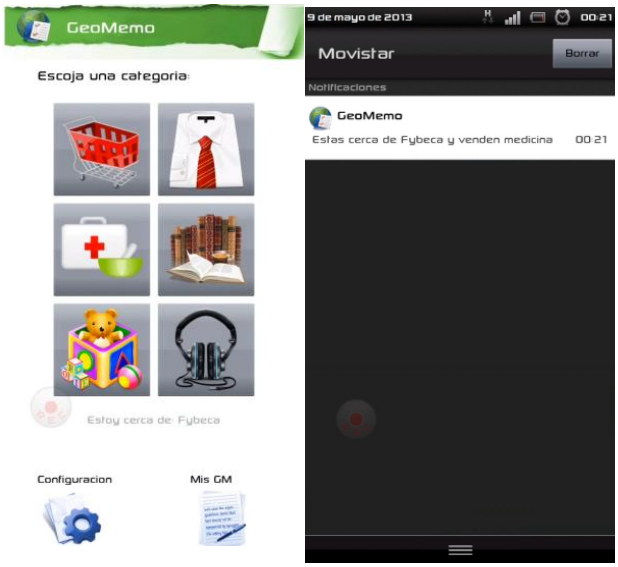


Figura 4.11. Probabilidad con la que la aplicación reconoce las tiendas.

Como podemos observar no siempre se reconocían las tiendas. En el caso de Fybeca se debía a que esta tienda tenía cerca diferentes redes que no provenían de un punto de acceso sino de dispositivos inteligentes que emitían señales WiFi. En el caso de Juguetón y Musicalísimo sucedía algo que habíamos previsto en la prueba anterior [Figura 4.7], estas dos tiendas se encontraban cerca, por ende tenían redes en común y rangos similares, cuando pasábamos cerca de las dos siempre se detectaba primero a Juguetón, esto también se debe a que nosotros implementamos una búsqueda lineal en nuestra base de datos. Es decir la velocidad con la que detectamos a las tiendas siempre dependerá al número total de tiendas en la base de datos.

Cabe destacar que la cantidad de mediciones para datos de WiFi dependerá de la velocidad con que el usuario camine, si camina lento (4 Km/h) detectara más redes WiFi y si camina a paso apresurado (6 Km/h) detectara menos redes. La frecuencia con que detecta los datos de GPS puede ser definida en el programa, en nuestro caso se hace cada cinco metros o cada minuto. Cada vez que se reconocía una tienda el nombre de esta era mostrado en pantalla [Figura 4.12].



(a) (b)

Figura 4.12. Aplicación al estar cerca de un local conocido (a) y al estar cerca del local deseado (b).

Pudimos notar que en la mayoría de casos las tiendas eran reconocidas en el Punto C debido a que este punto fue tomado como referencia, tal como se detalla en la prueba 4.3, la tabla VII nos muestra los resultados.

Tabla V. Puntos donde mayormente fueron reconocidas las tiendas

	A	B	C	D	E
Etafashion	x	x	x	x	
Fybeca		x	x		
Juguetón		x	x		x
Musicalísimo	x		x	x	
Megamaxi	x	x	x		x

También notamos que al estar dentro de un lugar cerrado como es el centro comercial las señales GPS no se detectaban correctamente, por ende la aplicación usaba más los datos del WiFi para detectar las tiendas. Gracias a la arquitectura de nuestro sistema podemos aprovechar esta situación y desactivar la detección de datos GPS, lo que ayudaría a tener un ahorro de energía en la batería del teléfono.

CONCLUSIONES

1. Se puede reutilizar el código escrito en C y C++ y llevarlo a Android NDK, obteniendo así menos tiempo de desarrollo y a su vez portarlo a otras plataformas.

2. Es posible usar varios procesos desarrollados en diferentes Frameworks (Android NDK y Android SDK) y crear como resultado un sistema que explote técnicas estándares de comunicación entre procesos así como técnicas de multi-hilos.
3. Los hilos creados usando la librería Pthreads en Linux corren de la misma manera que en Android NDK con mínimas modificaciones.

4. Definimos criterios de localización de personas u objetos dentro de un espacio cerrado.

5. La combinación de redes WiFi y señales GPS dan mejor precisión en localización al momento de realizar una búsqueda de personas u objetos.

6. El diseño de nuestra aplicación permite disminuir el consumo de energía en el momento de su ejecución ya que la fuente de datos (WiFi y GPS) son capaces de activarse o desactivarse.

RECOMENDACIONES

1. Para futuras investigaciones el proyecto debería ser probado en teléfonos con mejores prestaciones de hardware.

2. Si se desea realizar este proyecto a grandes escalas, la búsqueda sobre la base de datos o archivo de texto no debería ser de forma secuencial.

3. Nuestro proyecto también puede ser aplicado en entornos abiertos en donde se use más la información del GPS provista por el dispositivo móvil.

ANEXOS

Anexo A: Tabla de las características del Sistema Operativo Móvil Android

Tabla A.1. Características del sistema operativo Android.

Diseño de dispositivo	La plataforma es adaptable a pantallas de mayor resolución, VGA, biblioteca de gráficos 2D, biblioteca de gráficos 3D basada en las especificaciones de la OpenGL ES 2.0.
Almacenamiento	SQLite, una base de datos liviana, que es usada para propósitos de almacenamiento de datos.
Conectividad	Android soporta las siguientes tecnologías de conectividad: GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, HSDPA, HSPA+ y WiMAX.
Mensajería	SMS y MMS son formas de mensajería, incluyendo mensajería de texto y ahora la Android Cloud toDeviceMessaging Framework (C2DM) es parte del servicio de PushMessaging de Android.
Navegador web	El navegador web incluido en Android está basado en el motor de renderizado de código abierto WebKit, emparejado con el motor JavaScript V8 de Google Chrome.
Soporte de Java	Aunque la mayoría de las aplicaciones están escritas en Java, no hay una máquina virtual Java en la plataforma. El bytecode Java no es ejecutado, sino que primero se compila en un ejecutable Dalvik y corre en la Máquina Virtual Dalvik. Dalvik es una máquina virtual especializada, diseñada específicamente para Android y optimizada para dipositivos móviles que funcionan con batería y que tienen memoria y procesador limitados. El soporte para J2ME puede ser agregado mediante aplicaciones de terceros como el J2ME MIDP Runner.
Soporte multimedia	Android soporta los siguientes formatos multimedia: WebM, H.263, H.264 (en 3GP o MP4), MPEG-4 SP, AMR, AMR-WB (en un contenedor 3GP), AAC, HE-AAC (en contenedores MP4 o 3GP), MP3, MIDI, OggVorbis, WAV, JPEG, PNG, GIF y BMP.
	Streaming RTP/RTSP (3GPP PSS, ISMA), descarga progresiva de HTML (HTML5 <video>tag). Adobe Flash Streaming (RTMP) es soportado

Soporte para streaming	mediante el Adobe Flash Player. Se planea el soporte de Microsoft SmoothStreaming con el port de Silverlight a Android. Adobe Flash HTTP DynamicStreaming estará disponible mediante una actualización de Adobe Flash Player.
Soporte para hardware adicional	Android soporta cámaras de fotos, de vídeo, pantallas táctiles, GPS, acelerómetros, giroscopios, magnetómetros, sensores de proximidad y de presión,, sensores de luz, gamepad, termómetro, aceleración por GPU 2D y 3D.
Entorno de desarrollo	Incluye un emulador de dispositivos, herramientas para depuración de memoria y análisis del rendimiento del software. El entorno de desarrollo integrado es Eclipse (actualmente 3.4, 3.5 o 3.6) usando el plugin de Herramientas de Desarrollo de Android.
Google Play	Google Play es un catálogo de aplicaciones gratuitas o de pago en el que pueden ser descargadas e instaladas en dispositivos Android sin la necesidad de un PC.
Multi-táctil	Android tiene soporte nativo para pantallas capacitivas con soporte multi-táctil que inicialmente hicieron su aparición en dispositivos como el HTC Hero. La funcionalidad fue originalmente desactivada a nivel de *kernel* (posiblemente para evitar infringir patentes de otras compañías). Más tarde, Google publicó una actualización para el NexusOne y el Motorola Droid que activa el soporte multi-táctil de forma nativa.
Bluetooth	El envío de archivos (OPP) y la exploración del directorio telefónico, el marcado por voz junto con el envío de contactos entre teléfonos.
Video llamada	Android soporta videollamada a través de Google Talk desde su versión HoneyComb.
Multitarea	Multitarea real de aplicaciones está disponible, es decir, las aplicaciones que no estén ejecutándose en primer plano reciben ciclos de reloj, a diferencia de otros sistemas de la competencia en la que la multitarea es congelada (Como por ejemplo iOS, en el que la multitarea se limita a servicios internos del sistema y no a aplicaciones externas)
Características basadas en voz	La búsqueda en Google a través de voz está disponible como "Entrada de Búsqueda" desde la versión inicial del sistema.
Tethering	Android soporta tethering, que permite al teléfono ser usado como un punto de acceso alámbrico o inalámbrico.

Anexo B: Modificaciones en el Kernel

Se describen a continuación 2 modificaciones que realizo Google sobre el núcleo de Linux.

Binder

Android proporciona el modelo de componentes de proceso de unidades (*process-unit component model*), en el cual todos los componentes de Android son expresados como procesos de Linux en el espacio de usuario. Entre ellos encontramos a los servicios del sistema que permiten la ejecución de una aplicación, también a los que son responsables de la visualización de las aplicaciones en la pantalla, los que proveen las características de la cámara, entre otros [7].

Dado que Android se ejecuta basándose en la gestión de archivos, los procesos y memoria de Linux, los servicios del sistema de Android también son agrupados como procesos de Linux. Estos procesos se codifican tanto en Java como en C/C++, por lo que los procesos que se ejecutan en la Máquina virtual Dalvik, tales como WiFi, localización y activities, también son incluidos.

Para poder utilizar dispositivos móviles como teléfonos inteligentes, todas las funciones predeterminadas del sistema de Android (GPS, WiFi, Bluetooth, etc) se proveen como procesos del tipo servidor, siendo los clientes las aplicaciones que requieran de estas funciones. Por ejemplo, cuando una aplicación hace llamadas al

API de Java para obtener la información de ubicación (GPS) proporcionada por el SDK de Android, la aplicación envía un requerimiento al proceso de Linux que provee el servicio de localización interna y obtiene la respuesta [Figura B.1].

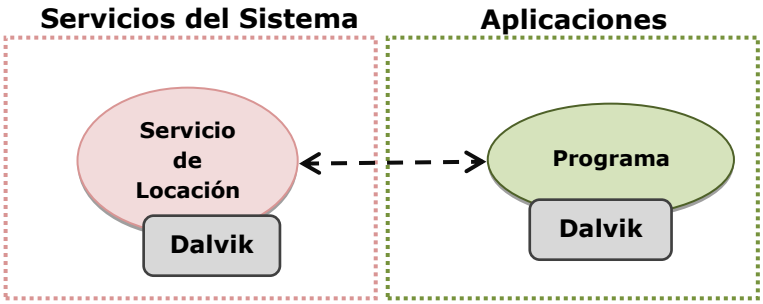


Figura B.1. Proceso de las aplicaciones que llama a los servicios de localización.

Administrador de energía

Un estándar de administración de la energía para las computadoras es ACPI (Interface de energía y configuración avanzada), que reemplaza a APM (Administrador de energía avanzada). Todos los equipos fabricados a partir del 2007 tienen soporte ACPI. ACPI maneja una estructura de árbol para apagar los dispositivos de manera ordenada y a diferencia de APM, ACPI permite el control de la administración de energía desde dentro del sistema operativo.

Anexo C: Guía de creaciones de aplicaciones en Android SDK.

Este anexo está basado en el artículo *Building Your First App* [11] el cual explica cómo realizar una aplicación para Android SDK. Cabe mencionar que la programación de aplicaciones utilizando el Android SDK se puede realizar en Windows, Mac o Linux pues hoy existen versiones para cada uno de estos sistemas operativos.

Todas las herramientas de desarrollo y los recursos oficiales de Android se orientan a desarrollar en el IDE Eclipse, este se puede descargar desde el sitio web de Eclipse Foundation.

Android SDK se encuentra disponible en la página de descargas del sitio web oficial Android Developers Guide seleccionando la versión correcta del sistema operativo. En Eclipse, instalamos el Android Developer Tools plugin agregando el siguiente repositorio: https://dl-ssl.google.com/android/eclipse/

Para cada aplicación Android que se quiera desarrollar se debe crear un nuevo proyecto. Cada vez que se crea un proyecto de Android, es necesario introducir los detalles de su aplicación y el nombre del paquete, así como la versión del SDK. Para crear los diseños de interfaz de usuario se usan archivos de formato XML. En las aplicaciones móviles las ventanas de los programas se les llama “Activities” por ende para cada pantalla dentro de la aplicación se debe crear un activity. La lógica del programa se la desarrolla usando lenguaje de programación Java.

Los detalles de la versión de la aplicación se los introduce en un archivo llamado AndroidManifest.xml. Aquí se puede añadir un número de versión y un nombre para que cada vez que se actualice la aplicación halla constancia de esto. Se puede probar y ejecutar las aplicaciones en dispositivos propios con Android, mientras estas se estén desarrollando. En Eclipse, también se puede usar el Android Virtual Device Manager para ejecutar las aplicaciones en emuladores en los cuales se puede especificar características de software y hardware.

De esta manera logramos un ambiente listo para desarrollar aplicaciones móviles para Android. En el sitio web de desarrolladores de Android podemos encontrar ejemplos sencillos para dar los primeros pasos en este entorno. [12]

Anexo D: Guía de instalación de Android NDK.

Nosotros realizamos el proyecto GeoMemo en la distribución Ubuntu 12.04, el IDE Eclipse Índigo (Versión 3.7). A continuación encontrara la guía de instalación de Android NDK que seguimos traducida al español, el enlace de la guía original lo encontrara en la bibliografía [20].

Paso 0: Descargar las herramientas

Usted necesita descargar el NDK. Vamos a hacer esto primero, ya que mientras se está descargando puede comprobar para asegurarse de que usted tiene las versiones correctas del resto de las herramientas que necesita.

Descargue el NDK para su sistema operativo desde el sitio Android.

Ahora, compruebe las versiones de sus herramientas con estas:

1. Si en Windows, Cygwin 1.7 o posterior
2. Actualización de awk a la versión más reciente (Windows)
3. GNU Make 3.81 o posterior (Linux)

Si cualquiera de estas versiones es demasiado antigua, favor, actualice antes de continuar.

Paso 1: Instalación del NDK

Ahora que el NDK se ha descargado, Es necesario descomprimirlo. Hágalo y colóquelo en un directorio apropiado. Se debe poner en el mismo directorio que pongamos el SDK de Android. Recuerde dónde lo puso.

En este punto, es posible que desee agregar las herramientas NDK a tu paso. Si estás en Mac o Linux, puede hacer esto con su configuración de la ruta original. Si estás en Windows usando Cygwin, es necesario configurar la ruta Cygwin.

Paso 2: Creación de un proyecto

Crear un proyecto regular de Android. Para evitar problemas en el futuro, el proyecto debe residir en una ruta que no contenga espacios. El proyecto tiene un nombre de paquete "com.mamlambo.sample.ndk1" con un nombre por defecto de actividad "AndroidNDK1SampleActivity" - ya verá estos aparecen más tarde.

En el nivel superior de este proyecto, cree un directorio llamado "jni" - este es el lugar donde poner tu código nativo. Si está familiarizado con JNI, el NDK Android se basa principalmente en los conceptos JNI - es, esencialmente, JNI con un conjunto limitado de cabeceras para compilar en C.

Paso 3: Agregando algo de código C

Ahora, dentro de la carpeta Jni, cree un archivo llamado native.c Coloque el siguiente código de C en este archivo para empezar, vamos a agregar otra función más tarde:

```
#include <jni.h>
#include <string.h>
#include <android/log.h>
#define DEBUG_TAG "NDK_AndroidNDK1SampleActivity"
```

```
void
Java_com_mamlambo_sample_ndk1_AndroidNDK1SampleActivity_helloLog(JNIE
nv * env, jobject this, jstringlogThis)
{
jbooleanisCopy;
const char * szLogThis = (*env)->GetStringUTFChars(env, logThis, &isCopy);
   __android_log_print(ANDROID_LOG_DEBUG,  DEBUG_TAG,  "NDK:LC:  [%s]",
szLogThis);
   (*env)->ReleaseStringUTFChars(env, logThis, szLogThis);
}
```

Esta función es en realidad bastante sencilla. Toma un parámetro String Java Object, la convierte en una cadena de lenguaje C, y luego la imprime en el Logcat.

El nombre de la función, sin embargo, es importante. Sigue el patrón específico de "Java": El nombre del paquete, seguido por el nombre de la clase, seguido del nombre del método, tal como se define a partir de Java. Cada pieza está separada por un guión bajo en lugar de un punto.

También son críticos los dos primeros parámetros de la función. El primer parámetro es el entorno JNI, utilizado con frecuencia con las funciones de ayuda. El segundo parámetro es el Java Object que es parte de esta función.

Paso 4: Llamadas Nativas desde Java

Ahora que ha escrito el código nativo, vamos a volver a Java. En la actividad

defecto, crear un botón y agregue un controlador de botón como quieras. Desde el interior de su controlador de botón, realice la llamada a helloLog:

```
helloLog("This will log to LogCat via the native call.");
```

Entonces usted tiene que añadir la declaración de la función en el lado de Java. Agregue la declaración siguiente a la clase de actividad:

```
private native void helloLog(String logThis);
```

Esto le dice a la recopilación y el sistema de conexión, que la aplicación de este método será desde el código nativo.

Por último, es necesario cargar la biblioteca que el código nativo en última instancia, a compilar. Agregue el inicializador estático a la clase de actividad para cargar la biblioteca por su nombre (el nombre de la biblioteca en sí depende de usted, y se hará referencia de nuevo en el paso siguiente):

```
static {
System.loadLibrary("ndk1");
}
```

Paso 5: Agregar el código nativo, archivo make

Dentro de la carpeta Jni, ahora tiene que añadir el makefile que se utilizará durante la compilación. Este archivo debe llamarse "Android.mk" y si usted designó a su native.c archivo y su librería ndk1, entonces el contenido Android.mk se verá así:

```
LOCAL_PATH:=$(call my-dir)
```

```
include $(CLEAR_VARS)
LOCAL_LDLIBS := -llog
LOCAL_MODULE := ndk1
LOCAL_SRC_FILES:= native.c
include $(BUILD_SHARED_LIBRARY)
```

Paso 6: Compilar el código nativo

Ahora que el código nativo está escrito y el archivo make está en su lugar, es el tiempo de compilar el código nativo. Desde la línea de comandos (los usuarios de Windows, tendrán que hacerlo dentro de Cygwin), tendrá que ejecutar el comando NDK-build desde el directorio raíz del proyecto. La herramienta NDK-build se encuentra dentro del directorio NDK, herramientas.

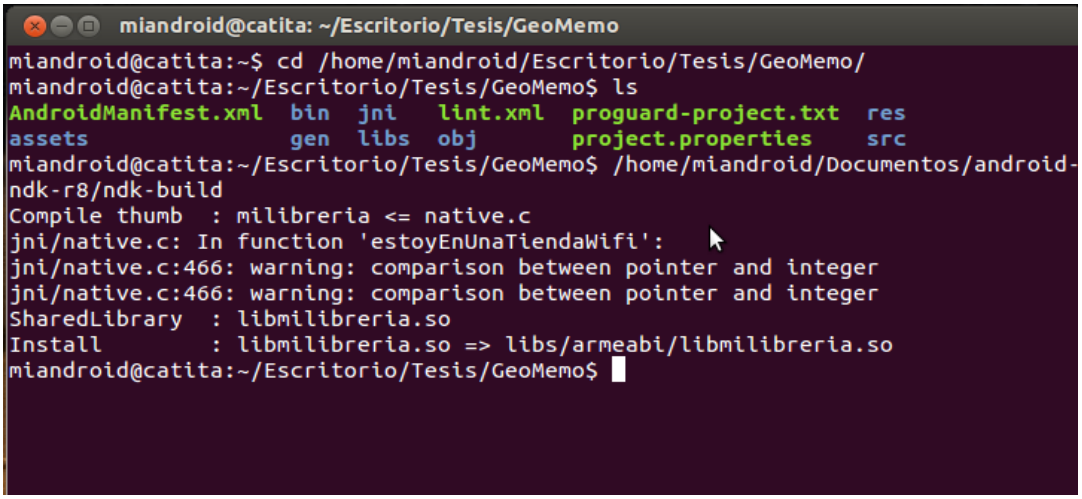


Figura D.1. Uso de ndk-build

La figura de la parte superior es la compilación de nuestro código del proyecto GeoMemo.

Por consiguiente se compila, puede asegurarse de que todo se vuelve a compilar si se utiliza el comando "NDK-buildclean".

Paso 7: Ejecución del Código

Ahora ya está todo listo para ejecutar el código. Cargue el proyecto en su emulador favorito, vea el Logcat, y presione el botón para ejecutarlo.

Una de dos cosas puede haber sucedido. En primer lugar, puede haber funcionado. Si es así, ¡felicitaciones! Sin embargo, es posible que desee leer, de todos modos. Usted probablemente tiene un error Logcat diciendo algo así como: "No se pudo ejecutar método de actividad." Esto está muy bien. Sólo significa que se ha perdido un paso. Esto es fácil de hacer en Eclipse. Por lo general, Eclipse está configurado para volver a compilar automáticamente. Lo que no hace es volver a compilar y vincular automáticamente si no sabe nada ha cambiado. Y, en este caso, lo que Eclipse no sabe es que ha compilado el código nativo. Por lo tanto, la fuerza de volver a compilar Eclipse por "limpiar" el proyecto (Proyecto-> Limpiar desde la barra de herramientas de Eclipse).

Acerca de los autores de la Guía de instalación

Desarrolladores Móviles Lauren Darcey Conder y Shane son co-autores de varios libros sobre el desarrollo de Android: un libro de programación a fondo titulado Android Desarrollo de Aplicaciones Inalámbricas *y* Sams Teach Yourself Desarrollo de aplicaciones Android en 24 horas*.* Desarrollan software para móviles de su empresa y prestan servicios de consultoría.

Anexo E: Diseño de la solución

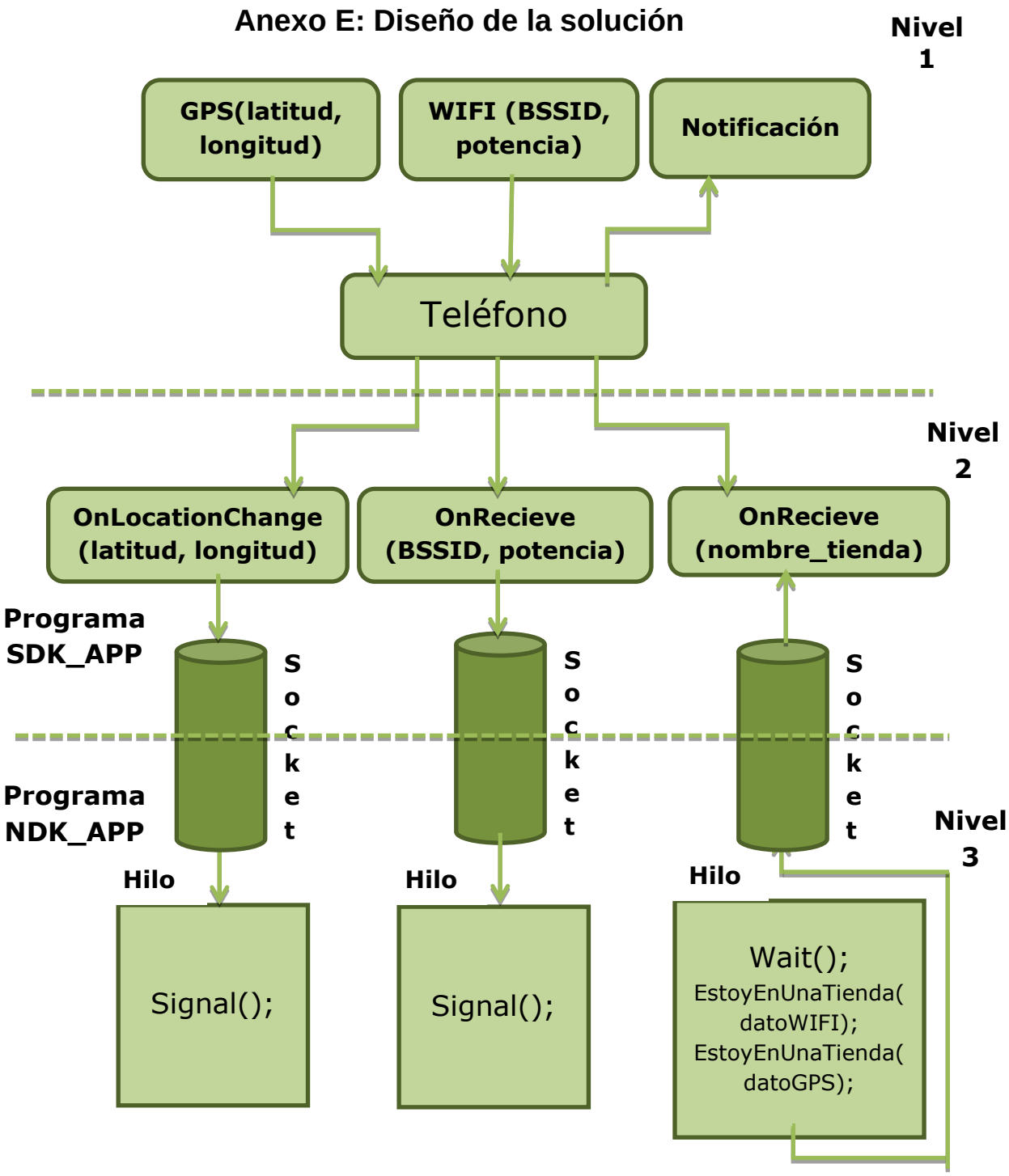
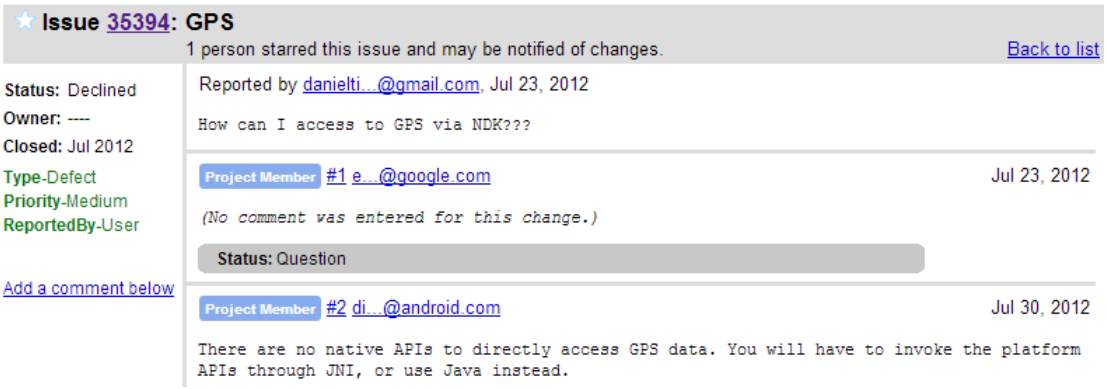


Figura E.1. Diseño de la solución.

Anexo F: Limitación de acceso al hardware en Android NDK

Respuesta de Google en el foro de desarrolladores a pregunta hecha por nosotros en el tema de acceso a bajo nivel de sensores.



Issue 35394: GPS

1 person starred this issue and may be notified of changes. Back to list

Status: Declined
Owner: ----
Closed: Jul 2012

Type-Defect
Priority-Medium
ReportedBy-User

Add a comment below

Reported by danielti...@gmail.com, Jul 23, 2012

```
How can I access to GPS via NDK???
```

Project Member #1 e...@google.com Jul 23, 2012

(No comment was entered for this change.)

Status: Question

Project Member #2 di...@android.com Jul 30, 2012

```
There are no native APIs to directly access GPS data. You will have to invoke the platform
APIs through JNI, or use Java instead.
```

Figura F.1. Respuesta en foro de Google

Anexo G: Archivo con formato kml generado en la aplicación web Google Maps.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml
xmlns="http://earth.google.com/kml/2.2">
<Document>
  <name>mall del sol tiendas</name>
<description><![CDATA[]]></description>
  <Placemark>
    <name>etafashion</name>
<description><![CDATA[]]></description>
    <styleUrl>#style6</styleUrl>
    <Polygon>
      <outerBoundaryIs>
        <LinearRing>
          <tessellate>1</tessellate>
          <coordinates>
            -79.892891,-2.155478,0.000000
            -79.892593,-2.155432,0.000000
            -79.892563,-2.155652,0.000000
            -79.892838,-2.155706,0.000000
            -79.892891,-2.155478,0.000000
          </coordinates>
        </LinearRing>
      </outerBoundaryIs>
    </Polygon>
  </Placemark>
  <Placemark>
    <name>fybecca</name>
<description><![CDATA[]]></description>
    <styleUrl>#style3</styleUrl>
    <Polygon>
      <outerBoundaryIs>
        <LinearRing>
          <tessellate>1</tessellate>
          <coordinates>
            -79.892944,-2.154832,0.000000
            -79.893074,-2.154853,0.000000
            -79.893211,-2.154631,0.000000
            -79.892990,-2.154510,0.000000
            -79.892944,-2.154832,0.000000
          </coordinates>
        </LinearRing>
      </outerBoundaryIs>
    </Polygon>
  </Placemark>
  <Placemark>
    <name>jugueton</name>
<description><![CDATA[]]></description>
    <styleUrl>#style1</styleUrl>
    <Polygon>
      <outerBoundaryIs>
        <LinearRing>
          <tessellate>1</tessellate>
          <coordinates>
            -79.891342,-2.155151,0.000000
            -79.891808,-2.155237,0.000000
            -79.891663,-2.155542,0.000000
            -79.891243,-2.155478,0.000000
            -79.891342,-2.155151,0.000000
          </coordinates>
        </LinearRing>
      </outerBoundaryIs>
    </Polygon>
  </Placemark>
  <Placemark>
    <name>musicalisimo</name>
<description><![CDATA[]]></description>
    <styleUrl>#style4</styleUrl>
    <Polygon>
      <outerBoundaryIs>
        <LinearRing>
          <tessellate>1</tessellate>
          <coordinates>
            -79.891602,-2.154920,0.000000
            -79.891579,-2.155030,0.000000
            -79.891487,-2.155017,0.000000
            -79.891510,-2.154904,0.000000
            -79.891602,-2.154920,0.000000
          </coordinates>
        </LinearRing>
      </outerBoundaryIs>
```

```
    </Polygon>
  </Placemark>
  <Placemark>
    <name>megamaxi</name>
<description><![CDATA[]]></description>
    <styleUrl>#style5</styleUrl>
    <Polygon>
      <outerBoundaryIs>
        <LinearRing>
          <tessellate>1</tessellate>
          <coordinates>
            -79.891342,-2.155046,0.000000
            -79.891197,-2.155014,0.000000
            -79.891174,-2.155111,0.000000
            -79.891319,-2.155137,0.000000
            -79.891342,-2.155046,0.000000
          </coordinates>
        </LinearRing>
      </outerBoundaryIs>
    </Polygon>
  </Placemark>
</Document>
</kml>
```

BIBLIOGRAFÌA

[1] INEC, “Reporte anual de estadísticas sobre tecnológicas de la información y comunicaciones TICs 2012”, http://www.ecuadorencifras.com, 2012.

[2] Jerry Hildenbrand, “Android A to Z: What is a kernel?”, http://www.androidcentral.com/android-z-what-kernel, Enero 23, 2012.

[3] “Android version History”, http://en.wikipedia.org/wiki/Android_version_history.

[4] Peter McDermott, “Porting Android to a new Device- What did Google change in the kernel?”, http://www.linuxfordevices.com/c/a/Linux-For-DevicesArticles/Porting-Android-to-a-new-device/, Diciembre 4, 2008.

[5] John Stultz, “Waking systems from suspend”, http://lwn.net/Articles/429925/, Marzo 2, 2011.

[6] Robert Martin, “An Introduction to Android Shared Memory”, http://notjustburritos.tumblr.com/post/21442138796/anintroduction-to-android-shared-memory, Junio, 2012.

[7] Ahn Joonseok, “Binder: Communication Mechanism of Android Processes”, http://www.cubrid.org/blog/dev-platform/bindercommunication-mechanism-of-android-processes/, Octubre , 2012.

[8] Frank Maker, “A Survey on Android vs. Linux”,http://handycodeworks.com/?p=10, Febrero 23, 2011.

[9] Huang Junwei & Wang Borong, “Power Management from Linux Kernel to Android”, http://goo.gl/eNx7d, 2010.

[10] Frank Maker, “Android Dalvik Virtual Machine”, http://jaxenter.com/android-dalvik-virtual-machine-35498.html#social, Abril 1, 2011.

[11] Google Android Developer, “Building Your First App”, http://developerlife.com/tutorials/?p=1375.

[12] Google Android Developer, “Samples”, http://developer.android.com/tools/samples/index.html.

[13] Fernando Cejas, “Android Location Providers - gps, network, passive”, http://developer.android.com/training/basics/firstapp/index.html, Octubre 28, 2010.

[14] Integrated Mapping Ltd., “How GPS Works”,http://www.maptoaster.com/maptoaster-topo-nz/articles/how-gpsworks/how-gps-works.html, 2009.

[15] Miguel Rueda Barranco, “Sockets: Comunicación entre procesos distribuidos”, http://es.tldp.org/Universitarios/seminario-2-sockets.html, Junio,1996.

[16] Mark Mitchell, Jeffrey Oldham and Alex Samuel, “Advanced Linux Programming” chap4-Threads, http://www.advancedlinuxprogramming.com/alpfolder/alp-ch04-threads.pdf, Junio, 2001.

[17] “Hilo de ejecución”, http://es.wikipedia.org/wiki/Hilo_de_ejecuci%C3%B3n, Marzo 9,2013.

[18] “Mapa del Mall del Sol”, http://developer.android.com/guide/topics/location/strategies.html, 2013.

[19] Google Android Developer, “Location Strategies”, http://malldelsol.com.ec/mapa, 2013.

[20] ShaneConder & Lauren Darcey, “Advanced Android: Getting Started with the NDK”, http://mobile.tutsplus.com/tutorials/android/ndk-tutorial/, Agosto 11, 2010.