Localización de personas en interiores por medio de redes WiFi usando el sistema operativo Android

Catalina Solís Miranda [(1)] Daniel Tigse Palma [(2)] Daniel Ochoa Donoso, PHD[(3)]
Facultad de Ingeniería en Electricidad y Computación [(1)]
Escuela Superior Politécnica del Litoral (ESPOL)
Campus Gustavo Galindo, Km 30.5 vía Perimetral
Apartado 09-01-5863. Guayaquil-Ecuador
csolis@fiec.espol.edu.ec[(1)] dtigse@fiec.espol.edu.ec[(2)] dochoa@espol.edu.ec[(3)]

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.

Palabras Claves: *Android, GPS, Localización, Redes WiFi, Kit de desarrollo, Hilos, Sockets.*

Abstract

This paper provides a practical way how to implement concurrent processing concepts such as the use of threads and synchronization and communication between processes on mobile devices. As implementation of the knowledge acquired was a project that involves the development of a mobile application for the Android operating system where the user in one of the application stores a list of items you want to buy in a mall. Our application gave notice to the user via a notification when this happened near a place where they traded any of the items that were on this list. The application was programmed in C language for data processing and system calls, Java language was used only for access to phone's sensors and user interface.

Keywords*: Android, GPS, Location, WiFi networks, Development Kits, Threads, Sockets*.

1. 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.

2. Generalidades

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

2.1 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.

2.2 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.

2.3 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.

2.4 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).

3. Marco teórico

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

3.1. 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].

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.

3.3. 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 [5]. El primero provee la ubicación mediante GPS, este proveedor determina la localización mediante satélites. Dependiendo de las condiciones, puede tomar de 30 segundos a dos minutos [6] para

devolver una posición. El segundo, es el proveedor de ubicación mediante red celular, este la 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.

3.4. 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.

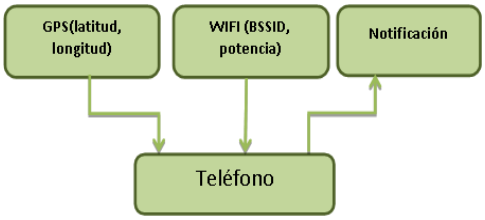
4. Implementación

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

4.1. Diseño de la solución

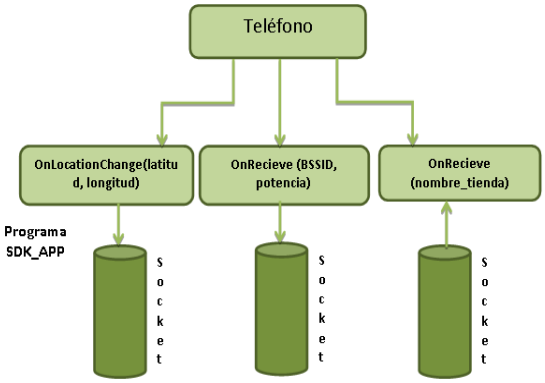
Basándonos en los conceptos previamente descritos diseñamos nuestro proyecto siguiendo el esquema de la Figura E.1. En esta se nombra al programa escrito en Java como programa SDK_APP y al programa escrito en C como programa NDK_APP.

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.



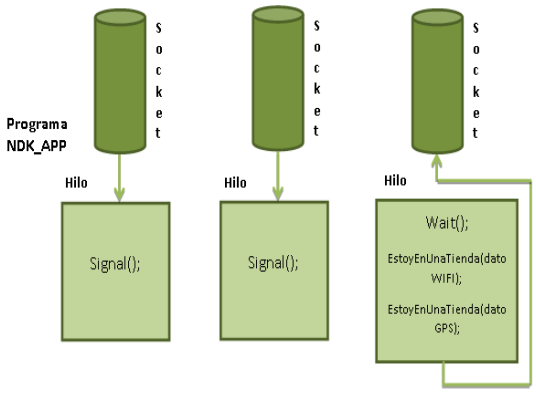
Nivel 1 Figura A.1.

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 A.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 A.1.

4.2. 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). La 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.

4.3. 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 [7], 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.

4.4. 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.

4.5. 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.

4.7. Hilos

Para la implementación de hilos en el programa NDK_APP se usó la librería GNU Pthread. La aplicación consta con tres hilos, cada uno aloja un socket para la comunicación con el programa SDK_APP. El primer hilo se encarga de alojar y mantener el socket por donde viaja la información del WiFi. El segundo hilo aloja y mantiene un socket que recibe información del GPS. 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 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.

4.8. 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.

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.

5. Resultados

5.1. 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.

5.1.1 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. 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.

5.1.2 Resultados obtenidos. 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.

Tanto en Android NDK como en Linux los hilos respondieron de la misma manera ya que en ambos programas 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.

5.2. 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.

5.2.1 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.

5.2.2 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. Cuando esto sucedía el cliente terminaba de ejecutarse y el resto del programa no funcionaba correctamente.

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 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.

5.3. 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.

5.3.1 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.

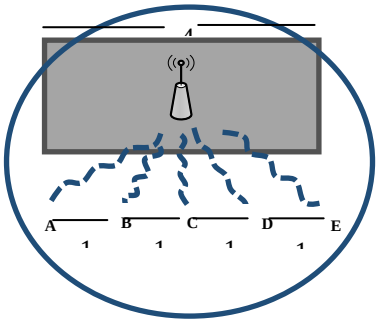


Figura 5.1. Ejemplo de un local con su propio enrutador.

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.

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.

5.3.2 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 5.2].

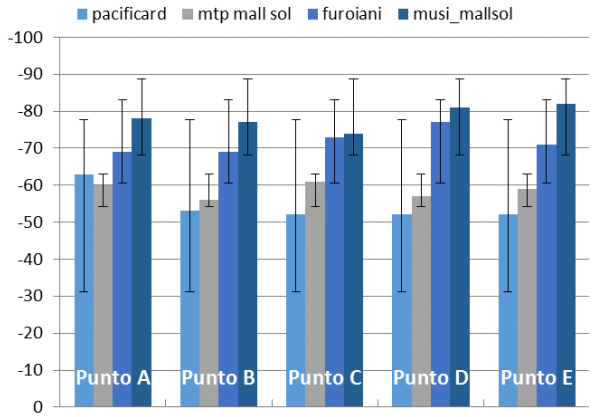


Figura 5.2. Niveles de potencia obtenidos y barras de error de una de las tiendas.

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. 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.

Pudimos también notar que en el caso de que dos o más tiendas estén cerca, 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. Los datos de latitud y longitud fueron obtenidos utilizando un mapa del centro comercial Mall del Sol [8] 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 y tratar que la posición de las áreas que creábamos sean las mismas.

Una vez que teníamos este mapa creado en Google Maps, lo exportamos a un formato kml, 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.

5.4. Prueba 4: Evaluación de proyecto

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

5.4.1 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.

5.4.2 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.

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, 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. 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 5.3, la tabla VII nos muestra los resultados. 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.

6. 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.

13. Agradecimientos

Agradecemos a Dios, a nuestras familias y a todas las personas involucradas en nuestro desarrollo profesional.

14. Referencias

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

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

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

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

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