Instituto Politécnico Nacional
Escuela Superior de Computo

Nombre del Alumno:

- Huerta Cortes Alan Antonio

Asignatura: Application Development For Mobile Devices

Profesor: Cifuentes Álvarez Alejandro Sigfrido

Reporte de la práctica Cliente Servidor

Fecha de entrega: 25 de febrero del 2019

Grupo: 3CV5

Índice

1.Objetivo:

Realizar un Programa en Midlet el cual comunique dos teléfonos por medio de bluetooth de tal forma de que uno de ellos funcione como servidor y el otro haga el papel de cliente, los cuales se podrán comunicar por medio de mensajes de texto proporcionados por los usuarios que se encuentren en el mismo canal de comunicación.

2.Conceptos previos

2.1Arquitectura Cliente-Servidor

Esta arquitectura consiste básicamente en un cliente que realiza peticiones a otro programa (el servidor) que le da respuesta. Aunque esta idea se puede aplicar a programas que se ejecutan sobre una sola computadora es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras. La interacción cliente-servidor es el soporte de la mayor parte de la comunicación por redes. Ayuda a comprender las bases sobre las que están construidos los algoritmos distribuidos.

Cliente: Programa ejecutable que participa activamente en el establecimiento de las conexiones. Envía una petición al servidor y se queda esperando por una respuesta. Su tiempo de vida es finito una vez que son servidas sus solicitudes, termina el trabajo.

Servidor: Es un programa que ofrece un servicio que se puede obtener en una red. Acepta la petición desde la red, realiza el servicio y devuelve el resultado al solicitante. Al ser posible implantarlo como aplicaciones de programas, puede ejecutarse en cualquier sistema donde exista TCP/IP y junto con otros programas de aplicación. El servidor comienza su ejecución antes de comenzar la interacción con el cliente. Su tiempo de vida o de interacción es “interminable”. Los servidores pueden ejecutar tareas sencillas (caso del servidor hora día que devuelve una respuesta) o complejas (caso del servidor ftp en el cual se deben realizar operaciones antes de devolver una respuesta). Los servidores sencillos procesan una petición a la vez (son secuenciales o interactivos), por lo que no revisan si ha llegado otra petición antes de enviar la respuesta de la anterior.

Los más complejos trabajan con peticiones concurrentes aun cuando una sola petición lleve mucho tiempo para ser servida (caso del servidor ftp que debe copiar un archivo en otra máquina). Son complejos pues tienen altos requerimientos de protección y autorización. Pueden leer archivos del sistema, mantenerse en línea y acceder a datos protegidos y a archivos de usuarios. No puede cumplir a ciegas las peticiones de los clientes, deben reforzar el acceso al sistema y las políticas de protección. Los servidores por lo general tienen dos partes:

Programa o proceso que es responsable de aceptar nuevas peticiones: Maestro o Padre.

Programas o procesos que deben manejar las peticiones individuales: Esclavos o Hijos.

2.2Bluetooth

El estándar bluetooth, del mismo modo que wifi, utiliza la técnica FHSS (Frequency Hopping Spread Spectrum, espectro ensanchado por saltos de frecuencia), que consiste en dividir la banda de frecuencia de 2.402 - 2.480 GHz en 79 canales (denominados "saltos") de 1 MHz de ancho cada uno y, después, transmitir la señal utilizando una secuencia de canales que sea conocida tanto para la estación emisora como para la receptora. Por lo tanto, al cambiar de canales con una frecuencia de 1600 veces por segundo, el estándar bluetooth puede evitar la interferencia con otras señales de radio.

El estándar bluetooth, del mismo modo que wifi, utiliza la técnica FHSS (Frequency Hopping Spread Spectrum, espectro ensanchado por saltos de frecuencia), que consiste en dividir la banda de frecuencia de 2.402 - 2.480 GHz en 79 canales (denominados "saltos") de 1 MHz de ancho cada uno y, después, transmitir la señal utilizando una secuencia de canales que sea conocida tanto para la estación emisora como para la receptora. Por lo tanto, al cambiar de canales con una frecuencia de 1600 veces por segundo, el estándar bluetooth puede evitar la interferencia con otras señales de radio.

¿Cómo se establecen las conexiones mediante 'bluetooth'?

El establecimiento de una conexión entre dos dispositivos bluetooth sigue un procedimiento relativamente complicado para asegurar un cierto nivel de seguridad: Modo pasivo > Solicitud > Búsqueda de puntos de acceso > Paginación > Sincronización con los puntos de acceso > Descubrimiento del servicio del punto de acceso > Creación de un canal con el punto de acceso > Emparejamiento mediante el PIN (seguridad) > Utilización de la red.

Durante el uso normal, un dispositivo funciona en "modo pasivo", es decir, que está escuchando la red. El establecimiento de una conexión comienza con una fase denominada solicitud, durante la cual el dispositivo maestro envía una solicitud a todos los dispositivos que encuentra dentro de su rango, denominados puntos de acceso. Todos los dispositivos que reciben la solicitud responden con su dirección. El dispositivo maestro elige una dirección y se sincroniza con el punto de acceso mediante una técnica denominada paginación, que principalmente consiste en la sincronización de su reloj y frecuencia con el punto de acceso.

De esta manera se establece un enlace con el punto de acceso que le permite al dispositivo maestro ingresar a una fase de descubrimiento del servicio del punto de acceso, mediante un protocolo denominado SDP (Service Discovery Protocol, protocolo de descubrimiento de servicios). Cuando esta fase de descubrimiento del servicio finaliza, el dispositivo maestro está preparado para crear un canal de comunicación con el punto de acceso, mediante el protocolo L2CAP.

2.3 J2ME y Bluetooth

Los dispositivos Bluetooth que implementen este API pueden permitir que múltiples aplicaciones se estén ejecutando concurrentemente. El BCC previene que una aplicación pueda perjudicar a otra. El BCC es un conjunto de capacidades que permiten al usuario o al OEM2 resolver peticiones conflictivas de aplicaciones definiendo unos valores específicos para ciertos parámetros de la pila Bluetooth. El BCC puede ser una aplicación nativa, una aplicación en un API separado, o sencillamente un grupo de parámetros fijados por el proveedor que no pueden ser cambiados por el usuario. Hay que destacar, que el BCC no es una clase o un interfaz definido en esta especificación, pero es una parte importante de su arquitectura de seguridad.

La pila Bluetooth es la responsable de controlar el dispositivo Bluetooth, por lo que es necesario inicializarla antes de hacer cualquier otra cosa. El proceso de inicialización consiste en un número de pasos cuyo propósito es dejar el dispositivo listo para la comunicación inalámbrica. Desafortunadamente, la especificación deja la implementación del BCC a los vendedores, y cada vendedor maneja la inicialización de una manera diferente. En un dispositivo puede haber una aplicación con un interfaz GUI, y en otra puede ser una serie de configuraciones que no pueden ser cambiados por el usuario. Un ejemplo sería.

Dado que los dispositivos inalámbricos son móviles, necesitan un mecanismo que permita encontrar, conectar, y obtener información sobre las características de dichos dispositivos. En este apartado, vamos a tratar como el API de Bluetooth permite realizar todas estas tareas.

Cualquier aplicación puede obtener una lista de dispositivos a los que es capaz de encontrar, usando, o bien startInquiry() (no bloqueante) o retrieveDevices() (bloqueante). startInquiry() requiere que la aplicación tenga especificado un listener,el cual es notificado cuando un nuevo dispositivo es encontrado después de haber lanzado un proceso de búsqueda. Por otra parte, si la aplicación no quiere esperar a descubrir dispositivos (o a ser descubierta por otro dispositivo) para comenzar, puede utilizar retrieveDevices(), que devuelve una lista de dispositivos encontrados en una búsqueda previa o bien unos que ya conozca por defecto.

3. Desarrollo

El desarrollo se compone en la codificación del servidor y el cliente para establecer la comunicación entre ellos por medio de bluetooth.

1. Primero se desarrolla el código del servidor al cual se conectará el cliente.

   A continuación, se muestra la declaración de las variables que se utilizaran para la parte visual y los comandos del midlet.

```
6   public class ServidorBT extends MIDlet implements Runnable, CommandListener {
7       private Form    f;
8       private Display d;
        private TextField tf = new TextField("Mensaje:", "", 40, TextField.ANY);
        private Command ce = new Command("Enviar",       Command.CANCEL, 2);
        private Command cl = new Command("Borrar texto",Command.SCREEN, 2);
        private Command cc = new Command("Conectar",     Command.SCREEN, 2);
        private Command cs = new Command("Salir",        Command.SCREEN, 2);
14      private InputStream     is;
15      private OutputStream    os;
        private static final UUID id = new UUID("F0E0D0C0B0A000908070605040302010", false);
17      private LocalDevice     ld;
18      private StreamConnectionNotifier scn;
19      private Thread          t;
```

   A continuación, se muestra el código necesario para levantar el servidor**.**

```
    public void run() {
32      f = new Form("Servidor: Conectado al cliente.");
33      f.append(tf);
34      f.addCommand(ce);
35      f.addCommand(cl);
36      f.addCommand(cs);
37      f.setCommandListener(this);
38      try {
39          ld = LocalDevice.getLocalDevice();
40          if (!ld.setDiscoverable(DiscoveryAgent.GIAC)) { }
41          scn = (StreamConnectionNotifier) Connector.open("btspp://localhost:" + id.toString() + ";name=tcg");
        } catch (Exception ex) {
43          f.append("Error: " + ex);
44      }
45      d = Display.getDisplay(this);
46      d.setCurrent(f);
47      while (true) {
            StreamConnection conn = null;
49          try {
50              conn = scn.acceptAndOpen();
            } catch (Exception e) {
52              f.append("Error: " + e);
53              continue;
54          }
55          try {
56              os = conn.openOutputStream();
57              is = conn.openInputStream();
                while (conn != null) {
59                  byte buffer[] = new byte[40];
60                  is.read(buffer);
61                  f.insert(1, new StringItem("Cliente:", new String(buffer)));
62                  d = Display.getDisplay(this);
```

2. Después se debe de codificar el cliente que se conectara con el servidor

   A continuación, se muestra la parte visual y los comando que tendrá este midlet.

```
9   public class ClienteBT extends MIDlet implements DiscoveryListener, Runnable, CommandListener {
10
11      private Form f;
12      private Display d;
13      private List l;
        private TextField tf = new TextField("Mensaje:", "", 40, TextField.ANY);
        private Command ce = new Command("Enviar", Command.CANCEL, 2);
        private Command cc = new Command("Conectar", Command.SCREEN, 2);
        private Command cl = new Command("Borrar texto", Command.SCREEN, 2);
        private Command cs = new Command("Salir", Command.SCREEN, 2);
        private Vector vdi = new Vector();
        private Vector vda = new Vector();
        private int id[] = new int[20];
22      private int[] attrSet;
        private UUID[] uuid = {new UUID("F0E0D0C0B0A000908070605040302010", false)};
24      private int dis;
25      private InputStream is;
26      private OutputStream os;
27      private Thread t;
```

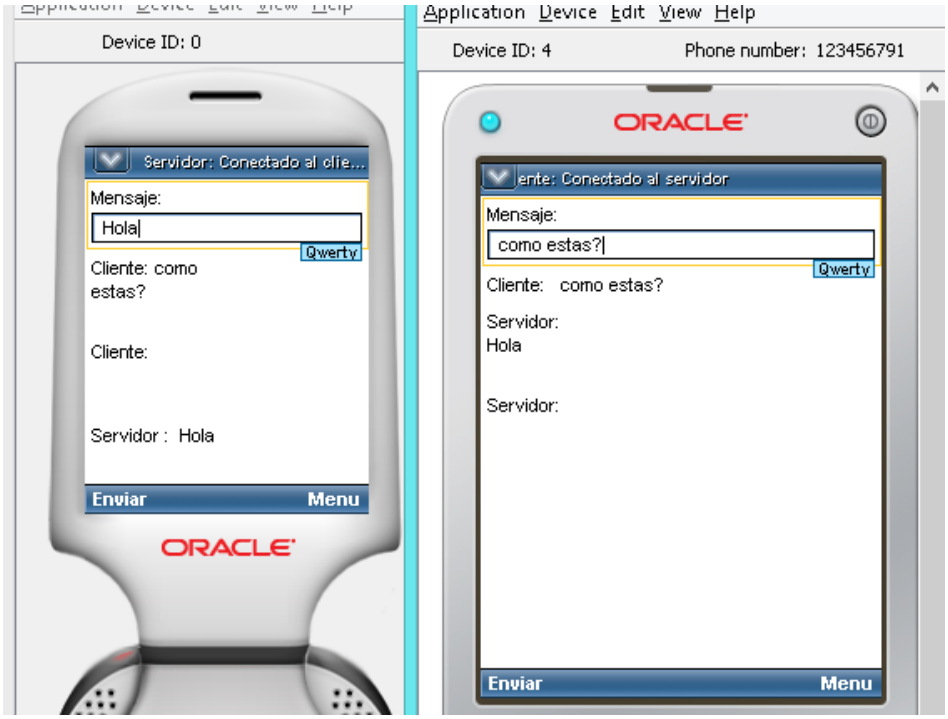
Código de inicio el cual busca que este un servidor a disposición.

```
    public void startApp() {
33      try {
34          d = Display.getDisplay(this);
35          l = new List("Servidores:", Choice.EXCLUSIVE);
36          l.addCommand(cc);
37          l.addCommand(cs);
38          l.setCommandListener(this);
39          d.setCurrent(l);
40          LocalDevice dispositivoLocal = LocalDevice.getLocalDevice();
41          if (!dispositivoLocal.setDiscoverable(DiscoveryAgent.GIAC)) {
42          }
43          DiscoveryAgent agenteDeBusqueda = dispositivoLocal.getDiscoveryAgent();
44          agenteDeBusqueda.startInquiry(DiscoveryAgent.GIAC, this);
45          while (vdi.size() == 0) {
46          }
47          for (int i = 0; i < vdi.size(); i++) {
48              try {
49                  id[i] = agenteDeBusqueda.searchServices(attrSet, uuid, ((RemoteDevice) vdi.elementAt(i)), this);
50                  l.append(((RemoteDevice) vdi.elementAt(i)).getFriendlyName(true), null);
                } catch (Exception e) {
52                  }
53          }
54          while (vdi.size() == 0 || vda.size() == 0) {
55          }
56          d.setCurrent(l);
        } catch (Exception e) {
58          }
59  }
```

Código que conecta con el servidor.

```
    public void run() {
        try {
            f = new Form("Cliente: Conectado al servidor");
            f.append(tf);
            f.addCommand(ce);
            f.addCommand(cl);
            f.addCommand(cs);
            f.setCommandListener(this);
            d = Display.getDisplay(this);
            d.setCurrent(f);
            String direccion = ((ServiceRecord) vda.elementAt(l.getSelectedIndex()))
            StreamConnection conn = (StreamConnection) Connector.open(direccion);
            is = conn.openInputStream();
            os = conn.openOutputStream();
            while (conn != null) {
                byte buffer[] = new byte[40];
                is.read(buffer);
                f.insert(1, new StringItem("Servidor:", new String(buffer)));
                d = Display.getDisplay(this);
                d.setCurrent(f);
            }
        } catch (Exception e) {
            f.append("Error : " + e);
        }
    }
```

Ejecución del cliente y servidor.



4.Conclusiones

Esta Practica se realizo con éxito, visualice el código necesario para conectar dos teléfonos con ayuda del bluetooth con una arquitectura cliente-servidor, lo cual me ayudara posteriormente a la realización del proyecto el cual consiste conectar el teléfono con un Arduino por medio del bluetooth y controlar el encendido y apagado de un led.

5.Bibliografia

https://www.ecured.cu/Arquitectura_Cliente_Servidor

http://bibing.us.es/proyectos/abreproy/11972/fichero/Cap%C3%ADtulo+4+-+J2ME+y+Bluetooth.pdf

https://es.ccm.net/contents/69-como-funciona-el-bluetooth

http://www.it.uc3m.es/celeste/docencia/j2me/tutoriales/bluetooth/EstudioTecnologico1_0.pdf