Universidad de Cartagena

Informe – DatagramSocket, DatagramPacket

Seminario de actualización

Amaury Rafael Ortega Camargo Y Ramiro Jose Verbel de la Rosa

Contenido

1 Objetivo

Comprender la comunicación de red usando las clases DatagramSocket y DatagramPacket alojadas en las librerías de java con el fin implementar un modelo cliente-servidor el cual proporcione instancias de servidores virtuales que contengan phpMyAdmin y MySQL accesibles mediante red por los clientes.

2 Librerías

Debido a que DatagramSocket y DatagramPacket se encuentran alojadas en las librerías de java, se hace uso de la librería mysql-connector-java-5.0.8-bin.jar la cual provee clases que son empleadas para la conexión a la base de datos encargada de registrar los servidores virtuales asignados a los clientes. Por lo tanto únicamente se presentan en la clase DB logrando el uso de bases de datos MySQL, esta clase se explica posteriormente en el documento. Específicamente las clases usadas de esta librería son:

2.1 com.mysql.jdbc.Driver

Esta clase es llamada para establecer una conexión con la base de datos, solo es instanciada una vez logrado que el driver dentro de la JVM esté listo para ser usado por medio del siguiente código:

```
Class.forName("com.mysql.jdbc.Driver");
```

2.2 com.mysql.jdbc.Connection

Esta clase es usada para instanciar y comenzar una conexión hacia una base de datos, especificando la dirección de red, el puerto y el nombre de la base de datos; además es necesario especificar las credenciales usando usuario y contraseña por medio del siguiente código:

```
Connection conexion = (Connection) DriverManager.getConnection(
        "jdbc:mysql://" + db_ip + ":" + db_port + "/" + db_name,
        user, pass);
```

Esta clase también nos permite preparar sentencias SQL destinadas a dicha conexión usando el método `createStatement.`

```
Statement st = (Statement) conexion.createStatement();
```

Al finalizar el uso de dicha conexión, es posible cerrarla usando el método `close().`

2.3 com.mysql.jdbc.Statement

Finalmente, esta clase nos permite ejecutar las sentencias SQL y en caso necesario retorna un objeto de tipo ResultSet. Del objeto es posible obtener el contenido de la DB por medio del nombre de las columnas usando el método getString (String). Esto junto nos permite ejecutar cualquier sentencia en nuestra base de datos por medio de la siguiente estructura:

```
// Saca el ultimo valor de la BD para tener los puertos
Query = "SELECT * FROM `registros`ORDER BY idUsuario DESC LIMIT 1";
Statement st = (Statement) conexion.createStatement();
resultSet = st.executeQuery(Query);
while (resultSet.next()) {
    usuario.setPuertoPHP(Integer.parseInt(resultSet.getString("puertoPHP")));
    usuario.setPuertoSQL(Integer.parseInt(resultSet.getString("puertoSQL")));
}
```

3 Clases

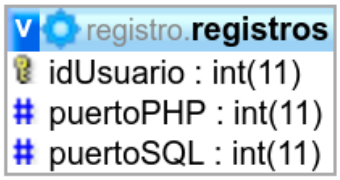
Debido a la necesidad de atender múltiples peticiones en el servidor usando hilos, se implementaron las siguientes clases:

3.1 Servidor

En la aplicación del servidor se tienen las siguientes clases:

3.1.1 DB

Permite manejar la base de datos usando la librería mencionada anteriormente y la clase Pc que será explicada posteriormente. La base datos usada está conformada por la siguiente tabla:



Esta clase nos permite trabajar con la base de datos por los siguientes métodos:

3.1.1.1 Pc insertar()

Genera una instancia de la clase Pc la cual contiene el id, puertoPHP y puertoSQL correspondientes al nuevo cliente pidiendo el servicio teniendo en cuenta los registros existentes en la base de datos.

3.1.1.2 void desconectar()

Detiene la conexión con la base de datos.

3.1.1.3 void eliminar(Integer id)

Elimina el registro en la base datos del usuario con el id proporcionado.

3.1.2 Pc

Clase para unificar la información de un cliente del servidor, contiene un `id`, `puertoPHP` y `puertoSQL`. Debido a la sencillez, esta clase solo tiene constructores y getters y setters.

3.1.3 Servidor

Esta clase es la encargada de instanciar DatagramSocket para escuchar nuevas peticiones de clientes y recibir el cliente mediante un objeto DatagramPacket, del cual se obtiene la información proporcionada por el cliente, para su optimo manejo estos objetos se envían a ServidorThread el cual se encarga de comunicarse con el cliente usando hilos permitiendo así que el servidor atienda a múltiples clientes simultáneamente. Para tratar con el Servidor se tienen los siguientes métodos:

3.1.3.1 *void iniciarServidor()*

Aquí se instancia el DatagramSocket encargado de aceptar las nuevas peticiones de los clientes.

```
// Crea el DatagramSocket para escuchar nuevas conexiones
servidorSocket = new DatagramSocket(puerto);
```

Luego se mantiene un ciclo infinito donde se recibe la conexión mediante un DatagramPacket el cual será equivalente al cliente, inmediatamente después se instancia un ServidorThread y se da inicio a un nuevo hilo el cual recibe como parámetros los objetos DatagramSocket y DatagramPacket.

```
while (true) {
    try {
        // Acepta una nueva conexion y crea un hilo para ella
        byte[] bufferIn = new byte[1000];
        clientePacket = new DatagramPacket(bufferIn, bufferIn.length);
        servidorSocket.receive(clientePacket);
        ((ServidorThread) new ServidorThread(clientePacket, servidorSocket)).start();
    } catch (IOException ex) {
        System.out.println("(LOG) [ERROR] No se pudo aceptar el cliente");
        Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

3.1.3.2 *void detenerServidor()*

Aquí se detiene el DatagramSocket dejando de escuchar nuevas peticiones de clientes, mediante el uso del método .close() de DatagramSocket.

3.1.4 ServidorThread

Esta clase trabaja en un hilo distinto al principal en la JVM por lo tanto el único método que tenemos es void run() el cual está siendo sobre-escribido proveniente de la clase madre Thread. Además, en el constructor de esta clase se crea una instancia de la clase DB, garantizando el uso de la base de datos solamente cuando sea necesario. Como extra se obtiene una instancia de la clase Runtime para crear y detener los servidores virtuales gestionados mediante docker.

3.1.4.1 *void run()*

Aquí se encuentra la lógica de la comunicación, obteniendo el mensaje del cliente recibido por el servidor usando el siguiente código:

```
mensajeRecibido = new String(clientePacket.getData());
```

Este servidor acepta 2 mensajes, “Dame server” o “Mata server[ID usuario]”. Dependiendo del mensaje, el servidor emplea la base de datos para insertar un nuevo usuario, crea el servidor virtual usando la instancia RunTime mencionada antes, luego responde al cliente mediante un DatagramPacket con la información de dicho usuario; o puede que el servidor identifique el ID del usuario que solicita destruir el servidor virtual, elimina su registro de la base de datos y procede a detener el servidor virtual. Sin importar el mensaje que reciba, se desconectará de la base de datos.

3.1.5 VentanaPrincipal

Esta clase forma la GUI del administrador encargado del servidor, la ventana cuenta con dos botones, los cuales tienen especificado sus métodos, iniciar y detener.

3.1.5.1 Inicio

Se crea un hilo donde se corre la instancia de la clase Servidor para escuchar nuevas peticiones.

```
t = new Thread(new Runnable() {
    public void run() {
        instancia.iniciarServidor();
    }
});
t.start();
```

3.1.5.2 Detener

Se detiene este hilo y la instancia de la clase Servidor.

```
t.stop();
instancia.detenerServidor();
```

3.2 Cliente

3.2.1 VentanaPrincipal

Esta clase es la GUI que usa el cliente para pedir el servicio del servidor, aquí se tienen 2 métodos que corresponden a 2 botones, Iniciar y Detener.

3.2.1.1 Inicio

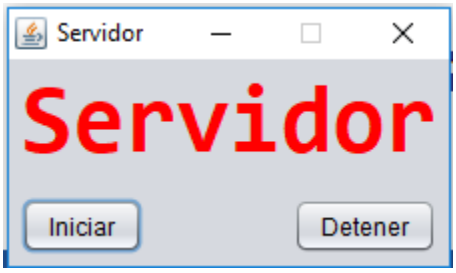
Se instancia un DatagramSocket que apunta al servidor por red, se envía el mensaje “Dame server” mediante un DatagramPacket y se espera la respuesta del servidor y se recibe mediante otro DatagramPacket, de la respuesta se obtiene el ID, IP del servidor, puertoPHP y puertoSQL. Con esto se arma una URL para acceder automáticamente al servicio de phpMyAdmin usando la clase Desktop de java la abre esta URL con el navegador predeterminado del cliente.

3.2.1.2 Detener

Aquí igual que en inicio, se instancia un DatagramSocket que apunta hacia el servidor por red, se envía el mensaje “Mata server[ID]” mediante un DatagramPacket, usando el id obtenido después de ejecutado el Inicio.

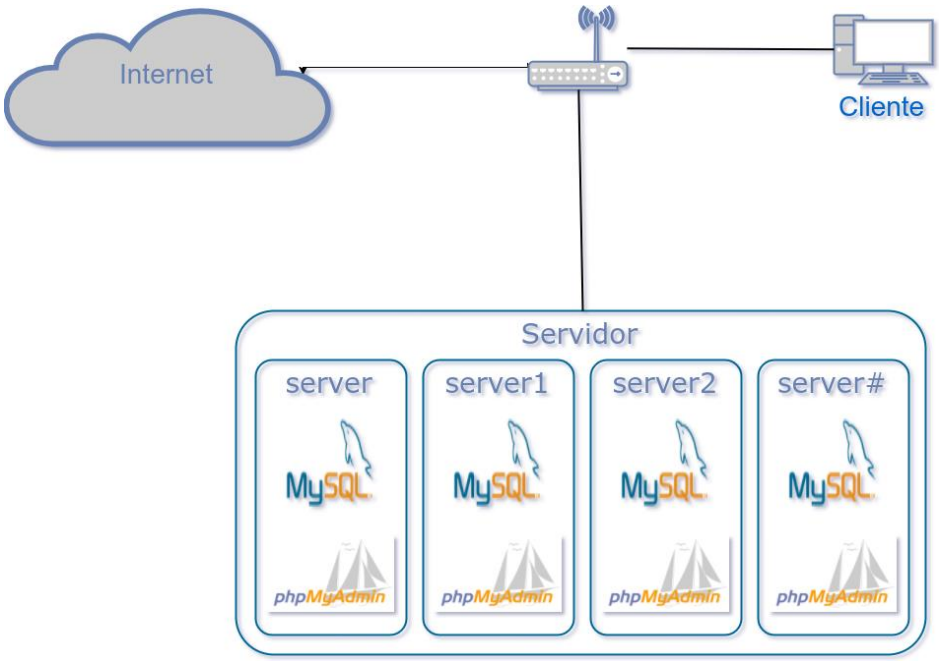
4 Implementación

Pantalla principal cliente



Pantalla principal servidor

4.1 Arquitectura



4.2 Pidiendo servidor

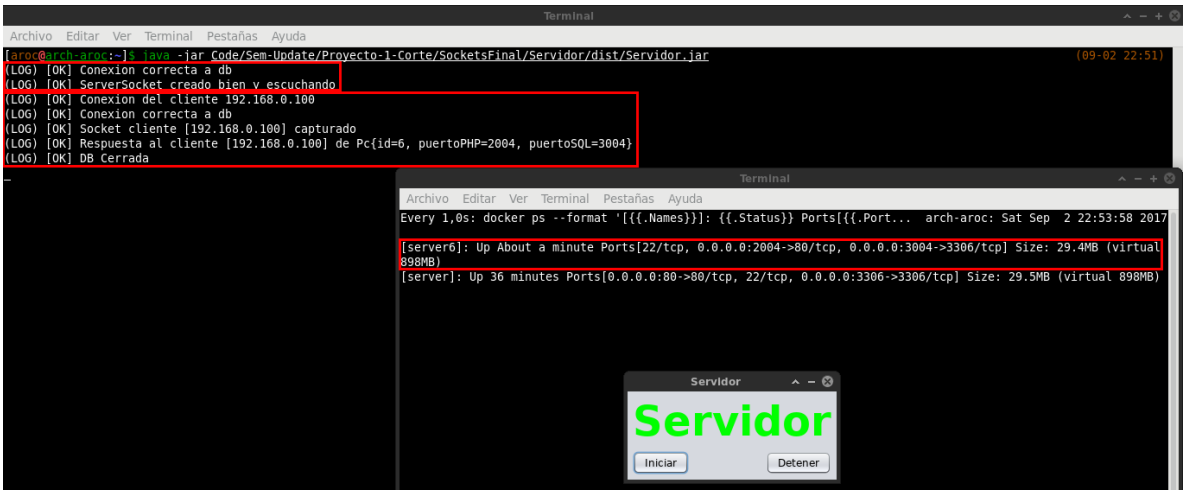
Teniendo el servidor iniciado y conectado a su base de datos correspondiente (Aquí ejecutamos la base de datos de registros dentro de docker llamada “server”).

Esto se logra al presionar el botón Iniciar lo cual crea un hilo donde se instancia un objeto de clase Servidor y es iniciado usando el método iniciarServidor(). Este método crea una instancia de DatagramSocket en un puerto determinado y luego entra en un ciclo infinito donde el servidor recibe las nuevas conexiones de los clientes usando el método .receive() el cual tiene como parámetro un DatagramPacket encargado de almacenar la información suministrada por el cliente.

Procedemos a hacer la petición del servicio desde el cliente usando el botón iniciar, lo cual crea la instancia de la clase DatagramSocket que apunta a la dirección en red del servidor y así enviando el mensaje “Dame server”.

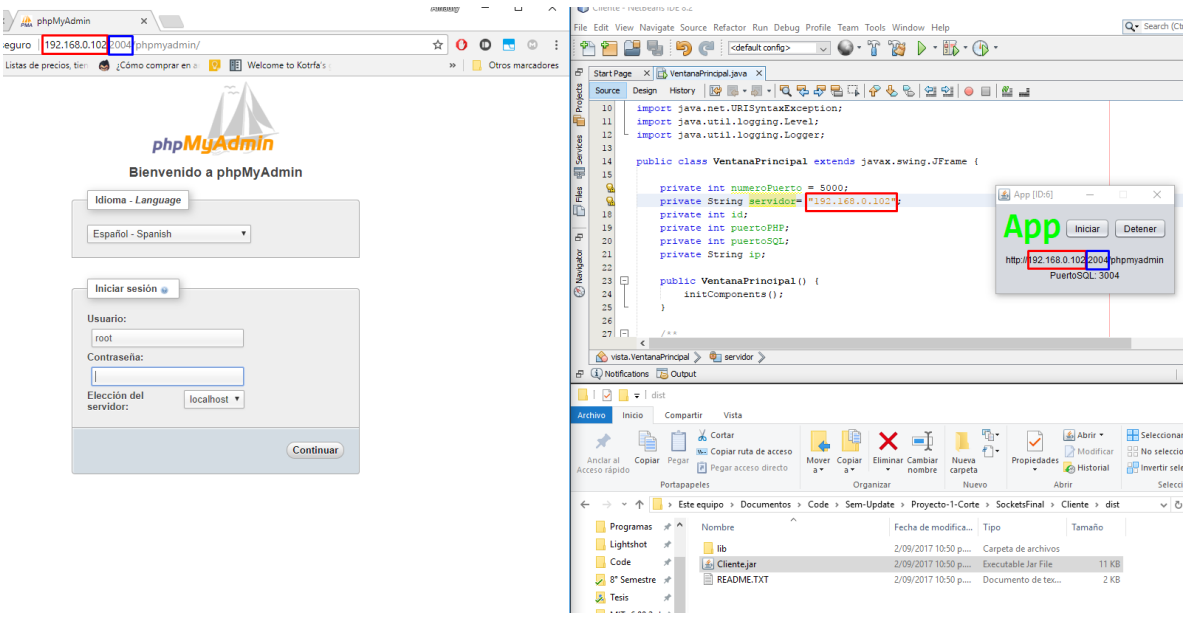
El servidor recibe la conexión con el método .receive() de la clase DatagramSocket el cual almacena el cliente en una objeto de tipo DatagramPacket, tanto el servidor como el cliente son pasados a una nueva instancia de la clase ServidorThread la cual no es asignada a ninguna variable, sin embargo, es ejecutada con el método .start().

El método start() ejecuta el método run() implementado en el servidor. Se captura el mensaje “Dame server” lo cual implica el uso de la base de datos mediante el método insertar() obteniendo una instancia de la clase Pc con id, puertoPHP y puertoSQL del nuevo cliente; seguido a esto se usa el método exec() de la clase RunTime el cual establece el servidor virtual del cliente usando la información de la instancia de la clase Pc. Este método exec() retorna una instancia de la clase Process lo que nos permite invocar el método waitFor() en dicha instancia el cual espera que el servidor termine el comando exec().



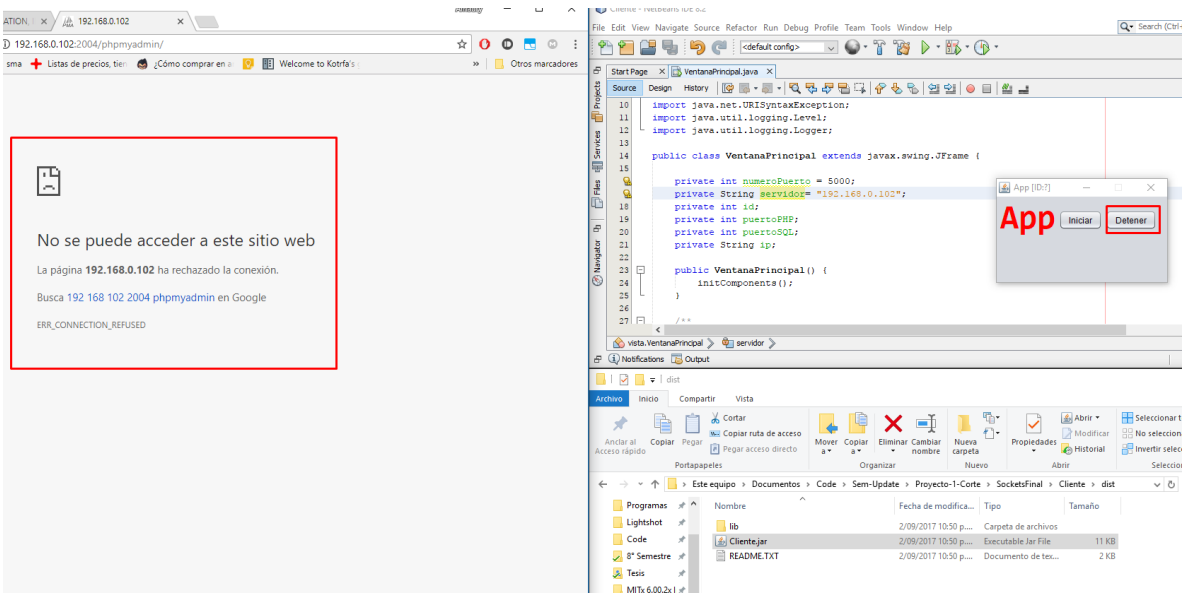
Al finalizar el servidor responde al cliente la información de la instancia de la clase Pc para que el cliente acceda a su servidor virtual. Finalmente, esta instancia de ServidorThread es desconectada de la base de datos y cerrada la comunicación con el cliente.

El cliente recibe la información de la mencionada instancia de la clase Pc, de la cual extrae la IP del servidor, ID del usuario, puertoPHP y puertoSQL. Con esto se arma una URL para que el cliente acceda a su servidor virtual, una vez armada la URL se usa el método Desktop.getDesktop().browse(URL) el cual ejecuta el navegador predeterminado del cliente y accede a su servidor virtual.



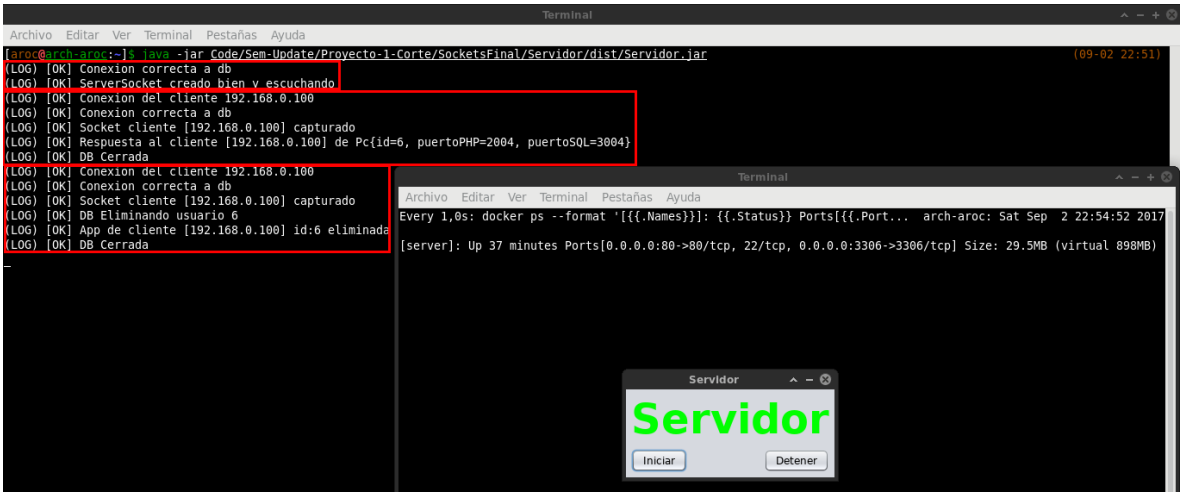
4.3 Deteniendo servidor

Luego de haber iniciado el servidor virtual, el cliente lo detiene usando el botón detener, esto encausa que se instancie la clase DatagramSocket hacia la dirección en red del servidor, mediante un DatagramPacket se logra comunicar el mensaje “Mata server[ID usuario]” al servidor, seguido a esto, se cierran las instancias creadas DatagramSocket.



El servidor recibe esta conexión en la instancia de la clase Servidor que le pasa dicha conexión a una nueva instancia de ServidorThread. Esto conlleva al método run() de ServidorThread, donde se obtiene el mensaje enviado por el cliente usando el DatagramPacket enviado desde la instancia de la clase Servidor. Se evalúa el mensaje usando una expresión regular y se extrae el ID para así eliminar el registro correspondiente en la base de datos usando una instancia de DB y el método eliminar(Integer), seguido a esto se hace uso de la instancia RunTime para ejecutar el comando que detiene el servidor virtual en el servidor.

Finalmente, ServidorThread procede desconectándose de la base de datos y cerrando los canales de comunicación con el cliente.



Posterior al uso, el administrador puede detener el escucha de conexiones usando el botón Detener y así para el hilo correspondiente a la instancia de la clase Servidor lo cual detiene el DatagramSocket responsable de escuchar nuevas peticiones.

5 Bibliografía

León Hernández, C. (s.f.). *Las clases Java DatagramPacket y DatagramSocket*. Obtenido de Universidad de La Laguna: http://nereida.deioc.ull.es/~cleon/doctorado/doc06/doc06/html/node8.html

ORACLE. (s.f.). *Class DatagramPacket.* Obtenido de Java™ Platform: https://docs.oracle.com/javase/7/docs/api/java/net/DatagramPacket.html

ORACLE. (s.f.). *Class DatagramSocket.* Obtenido de Java™ Platform: https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html

Ortega Camargo, A. R., & Verbel de la Rosa, R. J. (2017). *Repositorio Sem-Update.* Obtenido de Github: https://github.com/AmauryOrtega/Sem-Update

Telematica, D. d. (s.f.). *Trabajando con Sockets UDP*. Obtenido de Universidad Carlos III de Madrid: http://www.it.uc3m.es/celeste/docencia/cr/2003/PracticaSocketsUDP/