Universidad de Cartagena

Informe – Sockets

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 Socket y ServerSocket 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 `Socket` y `ServerSocket` provienen en java, solo usaremos la librería mysql-connector-java-5.0.8-bin.jar para hacer la conexión a la base de datos donde se llevara registro de que servidores virtuales están asignados. Debido a que estas clases son únicamente para tratar con bases de datos mysql, solo serán usadas en la clase DB explicada posteriormente. Las clases usadas son:

2.1 com.mysql.jdbc.Driver

Esta clase es llamada al momento de establecer una conexión con la base de datos, esta solo es instanciada una vez haciendo 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 llevar a cabo las sentencias SQL y en caso de ser necesario retorna un objeto de tipo `ResultSet`. Del `resultSet` es posible extraer el contenido 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

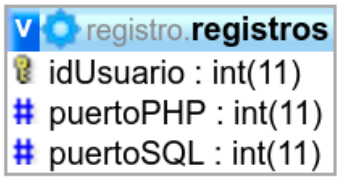
Debido a la necesidad de atender múltiples peticiones en el servidor usando hilos, se implementaron las siguientes clases:

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 `ServerSocket` y `Socket` para estar pendiente a nuevas peticiones de clientes y al aceptarlas, las envía 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 `ServerSocket` encargado de aceptar las nuevas peticiones de los clientes.

```
// Crea el ServerSocket para escuchar nuevas conexiones
servidor = new ServerSocket(puerto);
```

Luego se mantiene un ciclo infinito donde se acepta la conexión la cual retorna un `Socket` que será equivalente al cliente, inmediatamente después se instancia y comienza una instancia de `ServidorThread` y le pasamos el `Socket` de cliente.

```
while (true) {
    try {
        // Acepta una nueva conexion y crea un hilo para ella
        cliente = servidor.accept();
        ((ServidorThread) new ServidorThread(cliente)).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 `ServerSocket` para dejar de escuchar por nuevas peticiones de clientes usando el método `close()` de `ServerSocket`.

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-escribiendo proveniente de la clase madre `Thread`. Además, en el constructor de esta clase se crean instancias de las clases `ObjectInputStream` y `ObjectOutputStream` lo cual permitirán tener un flujo de datos hacia y desde el cliente. En este constructor también se crea una instancia de la clase DB y así garantizamos el uso de la base de datos solamente cuando es necesitado. Como extra se obtiene una instancia de la clase `Runtime` para poder crear y detener los servidores virtuales que se irán creando con docker.

3.1.4.1 *void run()*

Aquí se encuentra la lógica de la comunicación, primero se obtiene el mensaje que recibe el servidor usando el siguiente código:

```
mensajeRecibido = (String) in.readObject();
```

Este servidor acepta 2 mensajes, “Dame server” o “Mata server[ID usuario]”. Dependiendo del mensaje, el servidor usará la base de datos para insertar este nuevo usuario, crear el servidor virtual usando la instancia de `RunTime` mencionada antes y así responderle al cliente con la información de dicho usuario o el servidor identificará el ID del usuario que le solicita destruir su servidor virtual, eliminará su registro en la base de datos y procederá a detener el servidor virtual. Sin importar el mensaje que reciba, se desconectará de la base de datos y se cerraran las instancias de `ObjectInputStream`, `ObjectOutputStream` y `Sockets` creadas en el constructor.

3.1.5 VentanaPrincipal

Esta clase forma la GUI del administrador encargado del servidor, debido a que esta ventana es formada por 2 botones, se tendrá un método para cada botón, 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.

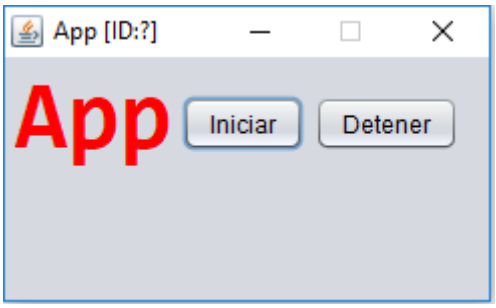
3.2.1.1 Inicio

Se instancia un `Socket` que apunta hacia el servidor por red y se crean dos instancias de las clases `ObjectInputStream` y `ObjectOutputStream` para así mantener una comunicación con el servidor. Aquí se envía el mensaje “Dame server” y se espera por la respuesta del servidor, 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 para abrir este URL con el navegador predeterminado del cliente.

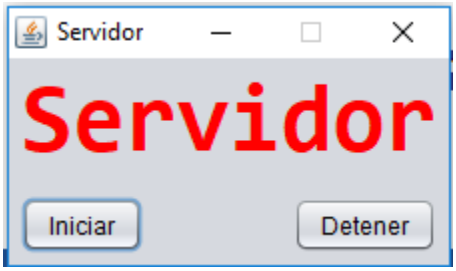
3.2.1.2 Detener

Aquí igual que en inicio, se instancia un Socket que apunta hacia el servidor por red y se crean dos instancias de las clases ObjectInputStream y ObjectOutputStream para así mantener una comunicación con el servidor. Aquí se envía el mensaje “Mata server[ID]” usando el id obtenido después de ejecutar `Inicio`.

4 Implementación

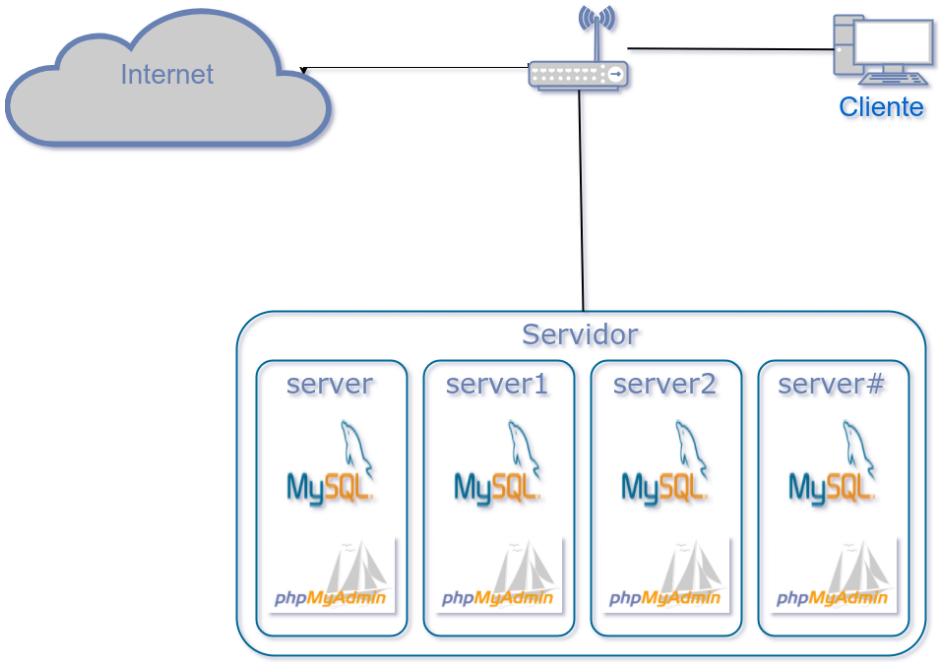


Pantalla principal cliente



Pantalla principal servidor

4.1 Arquitectura



4.2 Pidiendo servidor

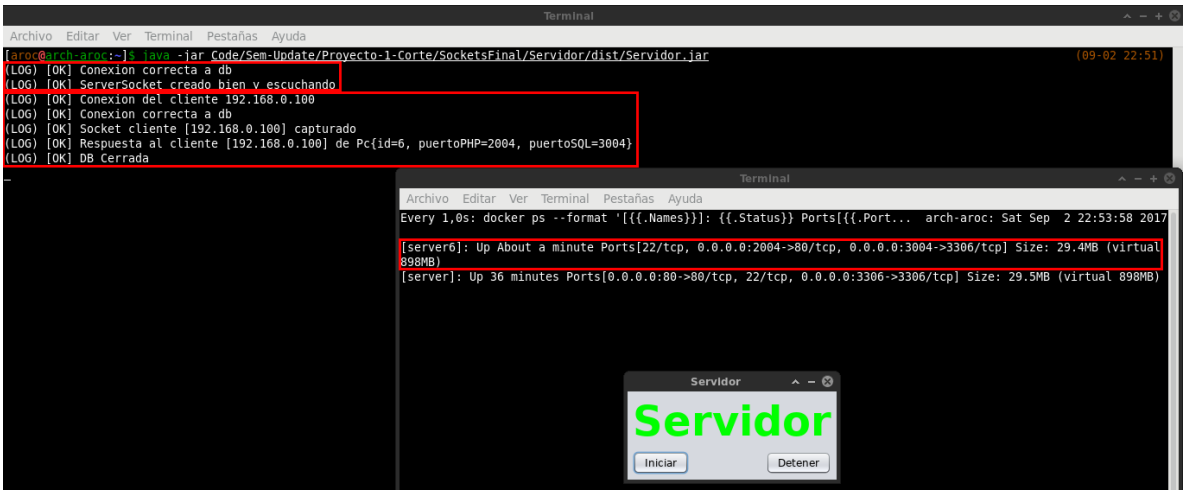
Teniendo el servidor iniciado y conectado a su base de datos correspondiente (Aquí ejecutamos la base de datos de registros dentro de docker llamada “server”).

Esto se logra presionando sobre el botón Iniciar lo cual creara un hilo donde una instancia de la clase Servidor es iniciada usando el método iniciarServidor(). Este método creará una instancia de

ServerSocket en un puerto determinado y luego entrará en un ciclo infinito donde se aceptaran las nuevas conexiones de los clientes usando el método accept() de la clase ServerSocket una vez haya alguna.

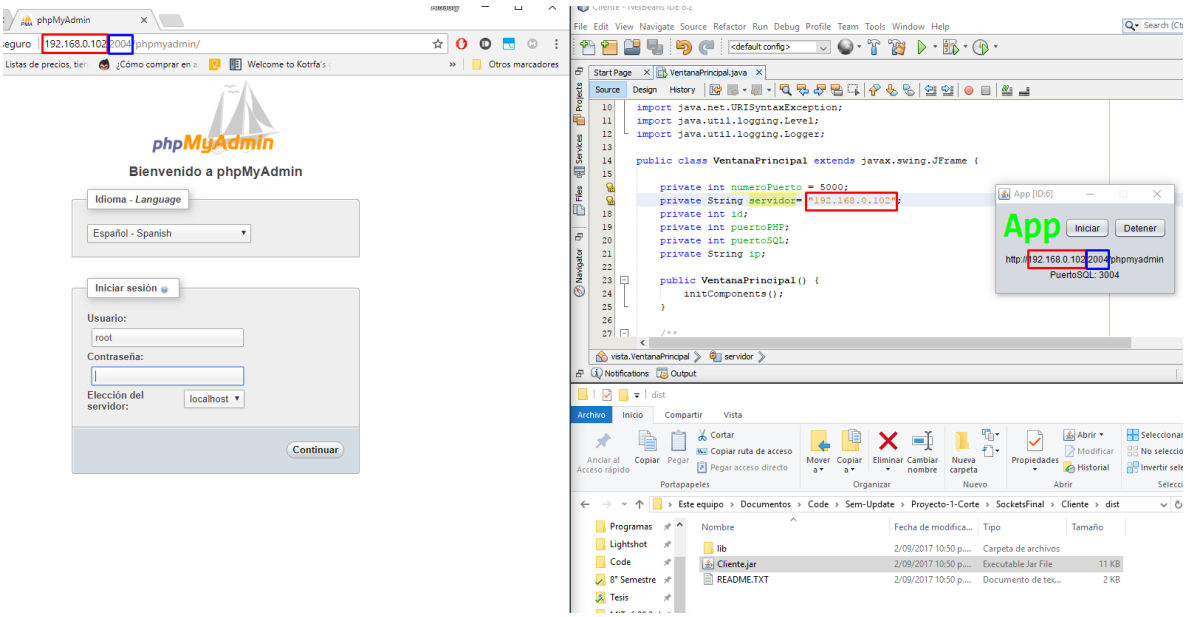
Procedemos a hacer la petición del servicio desde el cliente usando el botón iniciar, lo cual creará la instancia de la clase Socket que apuntará a la dirección en red del servidor y así le enviaremos el mensaje “Dame server”.

El servidor aceptará esta conexión con el método accept() de la clase ServerSocket el cual retornará una instancia de la clase Socket representando al cliente que será pasada a una nueva instancia de la clase ServidorThread la cual no será asignada a ninguna variable, sin embargo, si será ejecutada con el método start() de la clase ServidorThread. En este método start() se ejecutará el método run() que hemos implementado en el servidor. Aquí se capturará el mensaje “Dame server” lo cual implica que usará la base de datos con el método insertar() obteniendo así una instancia de la clase Pc con el id, puertoPHP y puertoSQL de este nuevo cliente; seguido a esto se usa el método exec() de la clase RunTime para ejecutar el servidor virtual del cliente usando la información de la instancia de la clase Pc. Este método exec() retornara una instancia de la clase Process lo que nos permitirá llamar el método waitFor() en esa instancia para esperar que el servido termine el comando exec().



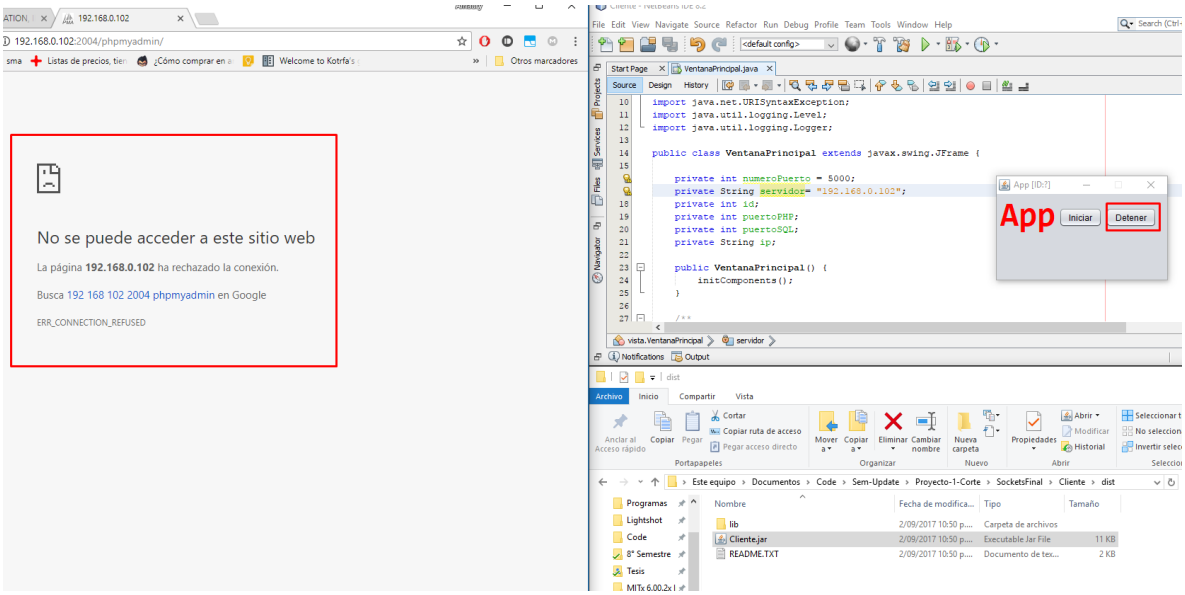
Al terminar esto el servidor le responde al cliente la información de la instancia de la clase Pc para que el cliente pueda acceder a su servidor virtual. Finalmente, esta instancia de ServidorThread se desconectará de la base de datos y cerrará la comunicación con el cliente.

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



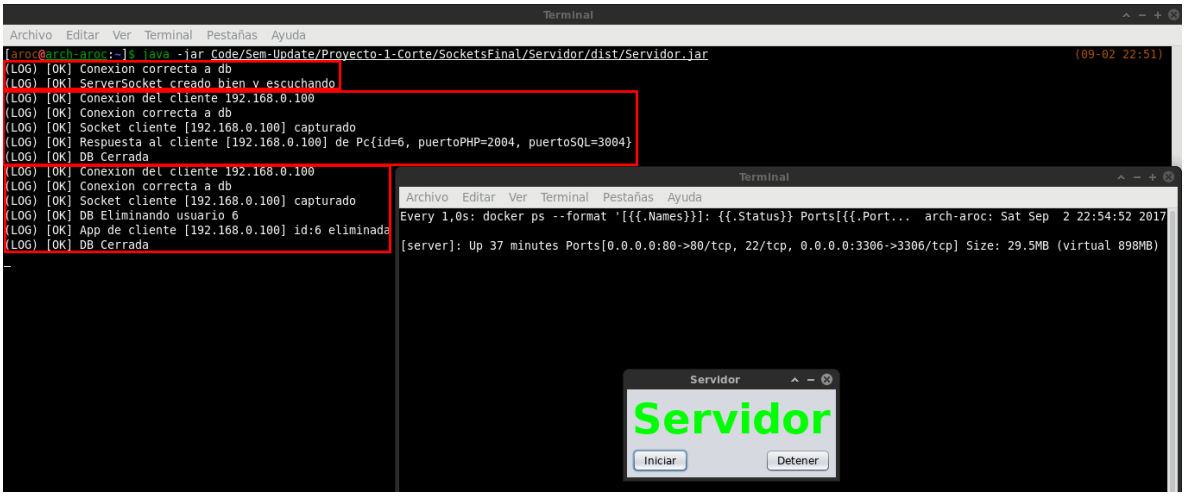
4.3 Deteniendo servidor

Una vez teniendo el servidor virtual corriendo, el cliente puede detenerlo usando el botón detener, esto causa que se instancie la clase Socket hacia la dirección en red del servidor junto a sus respectivos ObjectOutputStream y ObjectInputStream para lograr comunicarle al servidor el mensaje “Mata server[ID usuario]”, seguido a esto, se cierran las instancias creadas de ObjectOutputStream, ObjectInputStream y Socket.



El servidor escucha esta conexión en la instancia de la clase Servidor que le pasa dicha conexión a una nueva instancia de ServidorThread en forma de Socket. Esto lleva al método run de ServidorThread, aquí se obtiene el mensaje enviado por el cliente usando la instancia de Socket enviada desde la instancia de la clase Servidor. Se evalúa el mensaje usando una expresión regular y se extrae el ID para así proceder a eliminar su 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 ServerSocket responsable de escuchar por nuevas peticiones.

5 Bibliografía

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

ORACLE. (s.f.). *Lesson: All About Sockets.* Obtenido de ORACLE Java Documentation: https://docs.oracle.com/javase/tutorial/networking/sockets/index.html

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