PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

SISTEMA DE MENSAJES

NICOLÁS ALEXIS JOHNSON JOHNSON
FRANCO IGNACIO ORTIZ CABRERA
IGNACIO CRISTÓBAL JAMETT RODRÍGUEZ
CRISTÓBAL CÉSAR SÁNCHEZ CABALLERO

INFORME DE TAREA 1
PARA EL CURSO DE
REDES DE COMPUTADORES

OCTUBRE, 2018

Índice

1. Introducción

Un servidor es un equipo que escucha peticiones y devuelve una respuesta. Para esta aplicación, se aplicará un *Chat Server* que permiten intercambiar información a una gran cantidad de usuarios ofreciendo la posibilidad de llevar a cabo conversaciones en tiempo real. Precisamente en ésta aplicación, el cliente será el que las realice, generando así la comunicación Cliente-Servidor.

En este informe se abordarán los siguientes tópicos: Supuestos tomados, que es lo que creemos que debiera presentarse en el cliente y en el servidor, en el protocolo de nivel de aplicación usado, se explicará como se utilizaron los *sockets*. Además, la descripción de la situación actual, que muestra los requisitos solicitados, procesos involucrados, que es lo que se hace para generar la comunicación y el desarrollo de la aplicación, como se creó el *software* de mensajería mostrando y finalmente la explicación del código.

2. Supuestos tomados

Se asume que si todos los usuarios se desconectan, el *server* será apagado. Por último, se asume que el server siempre estará habilitado antes de que los clientes intenten conectarse.Se asumirá también que la dirección IP y el puerto de conexión serán siempre los mismos.

En la comunicación Cliente-Servidor, el cliente debe realiza una petición de envío de mensaje al servidor por medio de un *socket*. El servidor debe identificar las conexiones que tenga en un puerto para poder identificar la dirección a la cual debe ir el mensaje. Finalmente, el destino debe recibirlo para que el usuario pueda verlo.

Para el envío de mensajes entre cliente y servidor debe existir una comunicación establecida por medio de *sockets*. Para ello, el primero utilizará uno con la dirección y puerto del intermediario para establecer la conexión. El segundo, requiere dos para generar una conexión por medio de un puerto y otro para recibir datos del origen.

3. Protocolo de nivel aplicación usado

Se utilizaron *sockets* para realizar intercambio de datos entre dos programas. En cliente, se debe generar uno con la dirección IP y el puerto a utilizar, siendo éstos predeterminados. En el servidor se debe generar uno con el número del puerto en el que se realizará la comunicación y otro que reciba los mensajes.

Un *socket* es una estructura de datos abstracta que permite la conexión entre distintos procesos. En general, se utilizan para establecer comunicaciones entre distintas máquinas que estén conectadas a través de la red o incluso conexiones dentro de una misma máquina. En otras palabras, es un canal de comunicación entre dos programas que corren en uno o varios ordenadores.

4. Descripción de la situación actual

El *software* consiste en un *chat full duplex* con *sockets* desarrollado en lenguaje Java. Este consiste en un cliente que envía un mensaje a un servidor, este a su vez lo recibe y entrega la respuesta a otro cliente. Además la comunicación se realiza de manera simultánea, sin necesidad de esperar turnos.

5. Procesos involucrados

El proceso de este software, hecho en lenguaje Java, consta de la creación de una clase cliente, una clase servidor y una interfaz gráfica para cada una, con las que interactúan los usuarios. En el servidor se debe crear un *socket* asignándole un puerto para generar un enlace con un punto de comunicación; prepararse para escuchar y aceptar la conexión de un cliente (*accept()*); una vez realizada la conexión recibe el mensaje (*receive()*); se procesa y se envía una respuesta (*send()*); finalmente cuando termine la comunicación se cierra la conexión (*close()*). Por su parte, dentro de la clase cliente se crea un *socket* al cual se le asigna una dirección IP y un puerto para conectarse al servidor. Una vez hecho esto, se pueden enviar datos al servidor, recibir una respuesta del mismo y finalmente cuando termine la comunicación, cerrar la conexión.

6. Desarrollo de la aplicación

En este programa, las funciones que realiza simula una conversación de *Whatsapp*. En primer lugar, el cliente establece la conexión con el *server* mediante un *socket*, con su respectivo puerto y dirección IP. Luego, los pasos que sigue para ambos casos son:

- Envío de mensaje: Una vez establecida la conexión, aparecerá un aviso de çonexión establecida.[en] la interfaz gráfica. En la entrada de texto inferior, se debe escribir el texto y después presionar el botón enviar. Ejecutándose el código de envío. Luego, el *server* recibe el mensaje enviado por el cliente y para comprobar, en la ventana de éste, se mostrará lo entregado.

- Recibo de mensaje: Una vez que se ha enviado el mensaje al servidor, éste deberá reenviarlo al otro cliente conectado. Cuando llega dicho mensaje, el cliente lo recibirá para proceder a imprimirlo a la interfaz gráfica.

Una vez que el cliente no desee enviar y recibir mensajes, cierra la conexión con el *server*, para lo cual invoca el método *close()*. Para ambos actores, el envío y recibo de textos es similar, usando las funciones mencionadas.

El programa va a consistir de dos clases, la cual serán las realizarán la conexión Cliente-Servidor. Para este caso, ambos poseen interfaz gráfica, y las funciones necesarias para lograrlo. Estas son las siguientes:

- Clase Server_frame: Es la parte del servidor. En ella incluye la interfaz gráfica donde permitirá conectar para los clientes. Incluye un arreglo donde guardará los usuarios disponibles durante la conexión. También, incluirá la opción de desconectar.

- Clase Client_frame: Es la parte del cliente. En ella incluye la interfaz gráfica donde permitirá enviar mensajes al servidor. Además, le asignará un nombre por cada cliente, la cual se llamará *anonXXXX*, donde se asignará un número especifico al azar.

La conexión entre Cliente y Servidor se realizará en una dirección y puerto específico. En este caso, son *localhost* y 2222 respectivamente.