Tarea 3: Protocolo

Benjamín Pérez y Joaquín Fernandez

November 25, 2017

1 Descripción

Las aplicaciones de mensajería instantánea, tales como lo son Telegram, Whatsapp, Line, etc., han llegado a ser una parte importante de la vida diaria de la gente, debido a la facilidad de uso de éstas y a la comunicación "instantánea" que entregan. Nosotros no podemos quedar atrás, por lo que nuestro objetivo principal es lograr implementar un servicio de mensajería instantánea, pero de una manera más simple que los servicios existentes. Es decir, nos interesa lograr las dos funciones básicas de este tipo de servicio: envío de mensajes de texto y de archivos. Con esto, dejamos de lado el cifrado, la utilización de avatares, y otras funciones avanzadas.

El protocolo diseñado incorpora dos funciones básicas: La comunicación entre dos usuarios mediante mensajes de texto y el envío de archivos entre las partes respectivas. Además, los usuarios serán capaces de crear grupos entre ellos, y de enviar "Mulitcast", esto es, un mismo mensaje a una gran cantidad de otros usuarios.

Es importante notar que, para lograr este serivicio, nos concentraremos más en su confiabilidad que en su velocidad. Esto quiere decir que nos interesa que los mensajes lleguen efectivamente al usuario al cual están destinados, sin concentrarnos tanto en qué tan rápido llegan a él.

Todo esto será implementado en el lenguaje de programación C, debido a que es bacán.

2 Tipos de hosts

La comunicación se hará con formato cliente-servidor en ambos casos, tanto para una comunicación entre 2 usuarios, como para la comunicación a través de un grupo. Existe un solo servidor al cual todos los usuarios se conectan. A este servidor los usuarios envían "requests" para poder lograr la comunicación con otros usuarios, sea una comunicación privada, o grupal.

3 Formato de mensaje

El mensaje tendrá la siguiente estructura: [Usuario]: request. los requests serán manejados por el servidor de acuerdo a la información que se tenga del usuario. Por ejemplo, si el usuario está en un grupo, se esperarán dos posibles requests: "/exit" o "*", donde el primero sacará al usuario del grupo o chat privado, y el segundo, enviará un mensaje a todos los miembros del gupo. En caso de no estar en un chat, se esperan tres posibles requests: "/tg" (entrar al grupo) "/t [usuario]" (iniciar un chat privado con el usuario) y "/setname [nuevo nombre]" (cambia tu nombre de usuario a nuevo nombre).

4 Comunicación

Queremos que los mensajes de texto a enviar cumplan dos funciones: la primera, es que sean confiables, es decir, que siempre lleguen al usuario designado como el receptor, por lo que debemos manejar la pérdida de paquetes de tal manera que se reenvie un mensaje si se pierde en el camino, para que llegue en algún punto exitosamente al receptor. Esto se maneja automáticamente al usar las clases TcpClient y TcpListener, en la librería de Sockets. La segunda función a cumplir es poder enviar cualquier tipo de archivo, sea un texto, una imagen, un video, etc. Por lo tanto, un flujo de bytes confiable nos ayudará a cumplir dichas funciones. Al ser un flujo de bytes, nos aseguramos el envío de cualquier tipo de archivo deseado por un cierto emisor. Al ser confiable, existe un sistema de confirmación de que se recibió el mensaje, con lo que podremos manejar la falta de paquetes. Esto quiere decir que un cierto mensaje, si no logra llegar a su destino, será enviado nuevamente. Esto gracias a lo que se conoce como reenvío de mensajes con timeout, cuya función es, luego de cierto tiempo t, reenviar el mensaje si no se ha confirmado su llegada al destino.

5 Interacciones

Como mencionamos anteriormente, todos los clientes se conectarán a un servidor, el cual se encarga de mantener las conexiones entre usuarios, o la conexión de varios usuarios dentro de un grupo. Usando la clase TcpClient y TcpListener, podemos encargarnos de la confiabilidad de los mensajes. C nos entrega una API de sockets bastante completa, al simplificar el manejo de la confiabilidad. Es cosa de definir los clientes y el servidor como TcpClient y TcpListener para manejar la confiabilidad. En el diagrama de Cliente-Servidor, se puede ver que hay un usuario que actúa como servidor. Esto ya no es así, ahora cada cliente se conecta a un servidor, y este se conecta con el resto de los clientes.

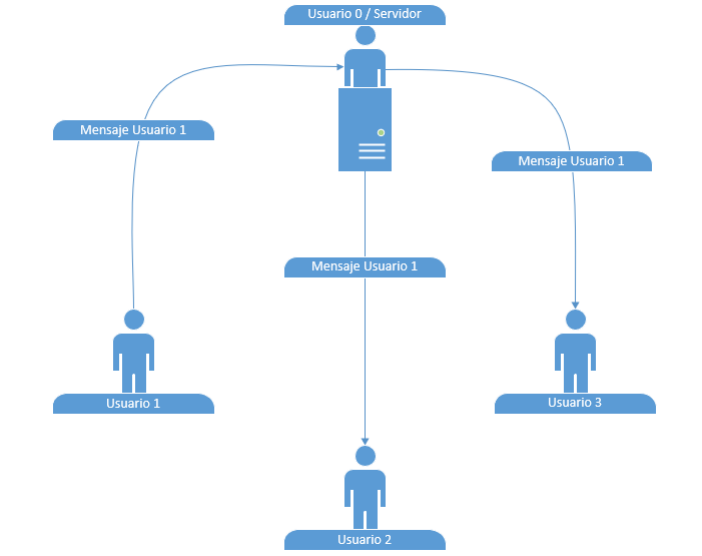

Figure 1: Diagrama Cliente-Servidor

6 Estados

Conectado Cuando un host hace login exitosamente y está disponible para empezar una nueva conversación.

Desconectado Cuando el host no está conectado al servidor.

Ocupado Cuando el host está conectado, pero está en una conversación con alguien más. En este estado todavía se puede invitar a este host a formar parte de otra conversación, pero de aceptar esa nueva llamada, sería desconectado del chat anterior y conectado a uno nuevo.