Sistemas Distribuidos

Grado en Ingeniería Informática

Universidad Carlos III de Madrid

Práctica:

Servicio de envío de mensajes Parte 1

Félix García Carballeira
Francisco Javier García Blas

8 de marzo de 2018

Índice

1. Objetivo

El objetivo de esta práctica es que el alumno llegue a conocer los principales conceptos relacionados con el diseño e implementación de una aplicación distribuida que utiliza distintas tecnologías (Sockets, RPC, Servicios Web SOAP y REST).

2. Descripción de la funcionalidad

El objetivo de la práctica es desarrollar un servicio de notificación de mensajes entre usuarios conectados a Internet, de forma parecida, aunque con una funcionalidad mucho más simplificada, a lo que ocurre con la aplicación WhatsApp. Se podrán enviar mensajes de texto de un tamaño máximo de 256 bytes (incluyendo el código 0 que indica fin de cadena, es decir, como mucho la cadena almacenada en el mensaje tendrá una longitud máxima de 255 caracteres) y de forma opcional se podrá también enviar archivos adjuntos de cualquier tamaño.

El esquema final de la aplicación es el que se muestra en la Figura 1.

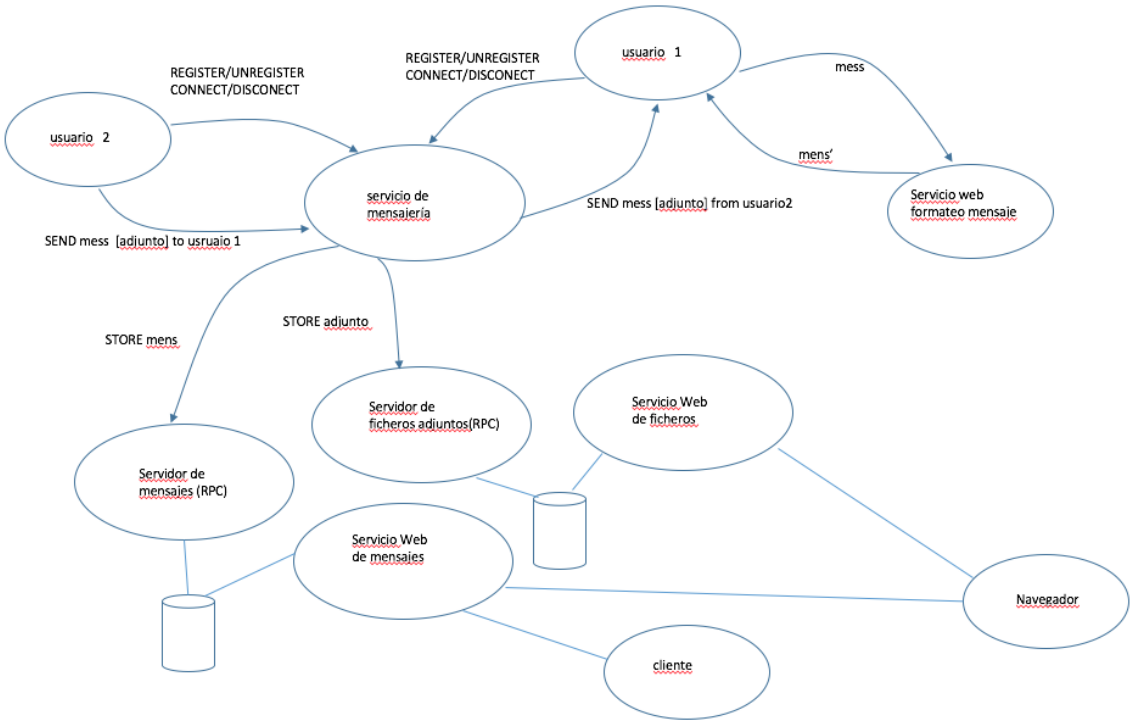

Figura 1: Servicio de notificación a desarrollar

El desarrollo de esta práctica se hará de forma incremental. En una primera parte, se deberá desarrollar la funcionalidad que se muestra en la figura 1. En esta primera parte no será necesario enviar ficheros adjuntos. Esta funcionalidad se incluirá más adelante.

3. Primera parte

El alumno deberá diseñar, codificar y probar, utilizando el lenguaje C y sobre un sistema operativo UNIX/Linux, un servidor que gestione el envío y recepción de mensajes entre los distintos clientes registrados en el sistema. Por otro lado, deberá diseñar, codificar y probar, utilizando el lenguaje Java y sobre un sistema operativo UNIX/Linux, un cliente que se comunique con el servidor y que permita el envío de mensajes a otros clientes así como la recepción de los mensajes enviados por otros clientes en el sistema.

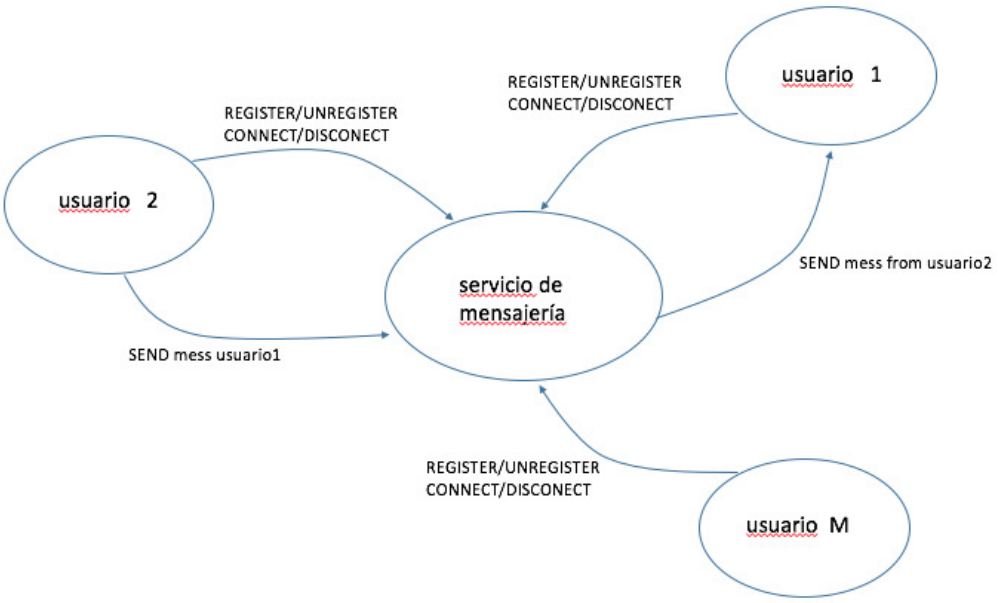

Figura 2: Servicio de notificación a desarrollar en la primera práctica

A continuación se detallan las características del sistema. En esta parte de la memoria se va a describir el protocolo a seguir entre el servidor y el cliente. Este protocolo permitirá a cualquier cliente que lo siga comunicarse con el servidor implementado. Esto hace que, diferentes alumnos puedan probar sus clientes con los servidores desarrollados por otros.

Para el almacenamiento de los usuarios y de los mensajes, en esta primera parte se podrán utilizar listas en memoria como las usadas en el primer ejercicio evaluable. Posteriormente, esta información habrá de almacenarse de forma persistente utilizando servicios que emplean RPC (llamadas a procedimientos remotos).

3.1. Desarrollo del servicio

El objetivo es diseñar y desarrollar los dos siguientes programas:

- Un **servidor concurrente multihilo** que proporciona el servicio de comunicación entre los distintos clientes registrados en el sistema, gestiona las conexiones de los mismos y el almacenamiento de los mensajes enviados a un cliente no conectado en el sistema.
- Un **cliente multihilo** que se comunica con el servidor y es capaz de enviar y recibir mensajes. Uno de los hilos se utilizará para enviar mensajes al servidor y el otro para recibirlos.

4. Desarrollo del cliente

El cliente ejecutará un intérprete de mandatos para comunicarse con el servidor siguiendo el mismo protocolo que éste.

El protocolo de comunicación se detalla en la sección 6.

4.1. Uso del cliente

Este intérprete se proporciona como material de apoyo.

Descomprima el archivo `materialApoyo-practica.tar`:

```
$tar xvf materialApoyo-practica.tar
```

Dentro de este directorio se encuentra el archivo *cliente.java* y todo el código necesario para su compilación. Para compilar este archivo y obtener su clase ejecutable ejecute:

```
$ javac   cliente.java
```

A continuación, puede ejecutar el programa cliente de la siguiente manera:

```
$ java -cp  .  client  -s <server> -p <port>
```

Donde **server** y **port** representan la dirección IP y el puerto del servidor. El nombre **server** puede ser tanto el nombre como la dirección IP del servidor. El intérprete de mandatos mostrará:

```
c>
```

y estará a la espera de que el usuario introduzca mandatos. Para finalizar este intérprete, el cliente debe escribir **QUIT**.

```
c> QUIT
```

A lo largo de la ejecución del cliente pueden obtenerse errores debidos a la red (servidor caído, fallo en la red, etc), todos estos errores hay que tenerlos en cuenta. Por tanto, se recomienda hacer un correcto tratamiento de los errores devueltos por las llamadas al sistema y de la biblioteca de sockets utilizada.

El cliente proporcionado incluye todo el código para leer los mandatos introducidos por el usuario y que se describen a continuación.

4.2. Registro en el sistema

Cuando un cliente quiera registrarse como usuario del servicio deberá ejecutar el siguiente mandato en la consola del cliente:

```
c> REGISTER <userName>
```

Este servicio una ve ejecutado en el servidor, puede devolver tres resultados (cuyos valores se describen detalladamente en la sección destinada a describir el protocolo de comunicación): 0 si la operación se ejecutó con éxito, 1 si ya existe un usuario registrado con el mismo nombre y 2 en cualquier otro caso. Si la operación se realiza correctamente, el cliente recibirá por parte del servidor un mensaje con código 0 y mostrará por consola el siguiente mensaje:

```
c> REGISTER OK
```

Si el usuario ya está registrado se mostrará:

```
c> USERNAME IN USE
```

En este caso el servidor no realizará ningún registro.

En caso de que no se pueda realizar la operación de registro, bien porque el servidor este caído o bien porque se devuelva el código 2, se mostrará en la consola el siguiente mensaje:

```
c> REGISTER FAIL
```

4.3. Darse de baja en el sistema

Cuando un cliente quiere darse de baja del servicio de mensajería deberá ejecutar el siguiente mandato en la consola:

```
c> UNREGISTER <userName>
```

Este servicio una vez ejecutado en el servidor, puede devolver tres resultados: 0 si la operación se realiza con éxito, 1 si el usuario no existe y 2 en cualquier otro caso. Si el usuario que se quiere dar de baja del servicio no existe, el servidor retornará un 1 (descrito en la sección destinada a describir el protocolo de comunicación) y se mostrará el siguiente mensaje por la consola del cliente:

```
c> USER DOES NOT EXIST
```

Si la operación de baja se realiza correctamente y el usuario es borrado en el servidor, el servidor devolverá un 0 y el cliente mostrará por la consola:

```
c> UNREGISTER OK
```

En caso de que no se pueda realizar esta operación, bien porque el servidor este caído y no se pueda establecer la conexión con él o bien porque devuelva el código 2, se mostrará en la consola el siguiente mensaje:

```
c> UNREGISTER FAIL
```

4.4. Conectarse al sistema

Una vez que un cliente está registrado en el sistema de mensajería, este puede conectarse y desconectarse del servicio tantas veces como desea. Para conectarse debe enviar (utilizando el protocolo descrito en la sección 5) al servidor su dirección IP y puerto para que éste pueda enviarle los mensajes de otros usuarios. La estructura de un proceso cliente conectado al servicio se muestra en la figura 3.

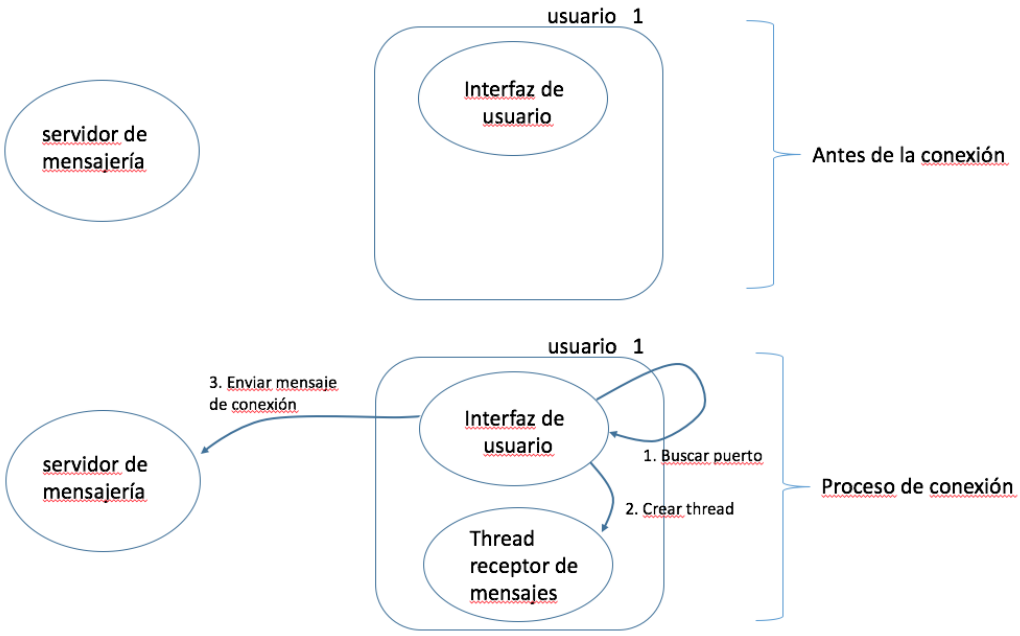

Figura 3: Estructura de un proceso cliente conectado al servicio de mensajería

Para ello el cliente introducirá por consola:

```
c> CONNECT <userName>
```

Internamente el cliente buscará un puerto válido libre (1). Una vez obtenido el puerto, y antes de enviar el mensaje al servidor, el cliente debe crear un hilo (2) que será el encargado de escuchar (en la IP y puerto seleccionado) y atender los envíos de mensajes de otros usuarios procedentes del servidor. A continuación el cliente enviará (3) la solicitud de conexión al servidor.

Una vez establecida la conexión en el sistema, el servidor devolverá un byte que codificará el resultado de la operación: 0 en caso de éxito, 1 si el usuario no existe, 2 si el usuario ya está conectado y 3 en cualquier otro caso.

Si todo ha ido bien, se mostrará por consola:

```
c> CONNECT  OK
```

En caso de código 1 (usuario no está registrado en el sistema), el cliente mostrará el siguiente error por consola:

```
c> CONNECT FAIL, USER DOES NOT EXIST
```

En caso de que el cliente ya estuviera conectado en el sistema (código 2), el cliente mostrará por la consola:

```
c> USER ALREADY  CONNECTED
```

En caso de que no se pueda realizar la operación de conexión, bien porque el servidor este caído, se produzca un error en las comunicaciones o se devuelva el código 3, se mostrará en la consola el siguiente mensaje:

```
c> CONNECT FAIL
```

4.4.1. Recepción de mensajes

Cada vez que el cliente reciba un mensaje a través del hilo creado para ello, deberá mostrar por consola el siguiente mensaje:

```
c> MESSAGE <id> FROM <userName>:
     <message>
     END
```

Como se verá en la sección de protocolo de comunicación, el mensaje de recepción llevará el remitente y el mensaje y un identificador (número entero) que lo identifica.

4.5. Desconectarse del sistema

Cuando un cliente no quiera seguir recibiendo mensajes del servidor (pero no quiera darse de baja del servicio), deberá ejecutar el siguiente mandato por consola:

```
c> DISCONNECT <userName>
```

Internamente el cliente debe parar la ejecución del hilo creado en la operación CONNECT. El servidor ante esta operación puede devolver 3 valores: 0 si se ejecutó con éxito, 1 si el usuario no existe, 2 si el usuario no está conectado y 3 en caso de error.

Si todo ha ido correctamente, el servidor devolverá un 0 y el cliente mostrará el siguiente mensaje por consola:

```
c> DISCONNECT OK
```

Si el usuario no existe, se mostrará el siguiente mensaje por consola:

```
c> DISCONNECT FAIL / USER DOES NOT EXIST
```

Si el usuario existe pero no se conectó previamente, se mostrará el siguiente mensaje por consola:

```
c> DISCONNECT FAIL / USER NOT CONNECTED
```

En caso de que no se pueda realizar la operación con el servidor, porque éste esté caído, hay un error en las comunicaciones o el servidor devuelve un 3, se mostrará por la consola el siguiente mensaje:

```
c> DISCONNECT FAIL
```

En caso de que se produzca un error en la desconexión, el cliente de igual forma parará la ejecución del hilo creado en la operación CONNECT, actuando a todos los efectos como si se hubiera realizado la desconexión.

4.6. Envío de un mensaje

Una vez que el cliente esté registrado y conectado al sistema podrá enviar mensajes a otros usuarios registrados. Para ello deberá ejecutar el siguiente mandato por la consola:

```
c> SEND <userName> <message>
```

Para implementar esta funcionalidad, el servidor asociará a cada mensaje enviado por un usuario un número entero como identificador y llevará siempre la pista de cuál ha sido el último identificador asignado a un mensaje de un usuario. Cuando un usuario se registra por primera vez en el sistema, este identificador se pone a 0, de forma que el primer mensaje que se envía toma como identificador el valor 1, el segundo el valor 2, y así sucesivamente. Cuando se llegue al máximo número de identificadores posibles, el nuevo identificador a asignar volverá a ser el 1, y se procederá de forma similar. El identificador debe almacenarse en una variable de tipo `unsigned int`, cuando se llegue al número máximo representable en una variable de este tipo y se le sume 1, la variable volverá a tomar valor 0 y se continuará el proceso, de forma que el siguiente identificador volverá a ser el 1.

Cuando se envía un mensaje al servidor, éste devuelve un byte con tres posibles valores (se describen con detalle en la sección 5): un 0 en caso de éxito, un 1 si el usuario no existe y 2 en cualquier otro caso. En caso de éxito (código 0), además devolverá el identificador asociado al mensaje enviado (un número entero) y se mostrará por consola el siguiente mensaje:

```
c> SEND OK - MESSAGE  id
```

En caso de que se envíe un mensaje a un usuario no registrado, el servidor indicará el error (código 1) y mostrará por consola:

```
c> SEND FAIL / USER DOES NOT EXIST
```

En caso de que se produzca un error (servidor caído, error de comunicaciones, error por problemas de almacenamiento de mensaje o se devuelva un error de tipo 2) se mostrará por consola:

```
c> SEND FAIL
```

Una vez que el servidor almacena un mensaje para un usuario y ha respondido con el código correspondiente al usuario remitente, si el usuario está conectado en ese momento le enviará el mensaje. En caso de que se haya enviado con éxito, el servidor enviará al remitente del mensaje la confirmación de que el mensaje con el identificador asignado se ha enviado al usuario correctamente (se describe con detalle en la sección 5). Cada vez que un cliente remitente de un mensaje recibe del servidor un mensaje de entrega de mensaje a otro proceso mostrará por consola:

```
c> SEND MESSAGE id OK
```

Indicando que el mensaje con identificador `id` se ha entregado correctamente.

En caso de que el usuario no esté conectado, el servidor almacenará el mensaje. Posteriormente cuando el cliente destinatario se conecte el servidor se encargará de enviarle todos los mensajes pendientes (uno a uno). Cada vez que se envía con éxito un mensaje a un usuario, se notifica el remitente del mensaje, el cual mostrará por pantalla:

```
c> SEND MESSAGE id OK
```

Como se verá posteriormente, siempre que el servidor envía con éxito un mensaje a un usuario, descarta el mensaje eliminándolo del servidor. Es decir, el servidor solo almacena los mensajes pendientes de entrega, cada vez que se entrega con éxito un mensaje se borra del servidor.

5. Desarrollo del servidor

El objetivo del servidor es ofrecer un servicio de comunicación entre clientes. Para ello los clientes deberán registrarse con un nombre determinado en el sistema y a continuación conectarse, indicando para ello su IP y puerto. El servidor debe mantener una lista con todos los clientes registrados, el nombre, estado y dirección de los mismos, así como una lista de los mensajes pendientes de entrega a cada cliente. Además se encargará de asociar un identificador a cada mensaje recibido de un cliente.

El servidor debe ser capaz de gestionar varias conexiones simultáneamente (debe ser concurrente) mediante el uso de múltiples hilos (multithread). El servidor utilizará sockets TCP orientados a conexión.

5.1. Uso del servidor

Se ejecutará de la siguiente manera:

```
$ ./server -p <port>
```

Al iniciar el servidor se mostrará el siguiente mensaje:

```
s> init server <local IP>:<port>
```

Antes de recibir comandos por parte de los clientes mostrará:

```
s>
```

El programa terminará al recibir una señal SIGINT (Ctrl+c).

5.2. Registro de un cliente

Cuando un cliente quiera registrarse enviará el mensaje correspondiente indicando el nombre de usuario. Cuando este mensaje es recibido, el servidor deberá hacer lo siguiente:

- Verificar que no existe ningún otro usuario registrado.
- Si no existe el usuario, se almacena la información con el nombre del usuario y se envía el código 0 al cliente. Se pone a 0 el valor asociado al identificador de mensaje.
- Si existe, se envía una notificación al cliente indicándolo.

La información asociada a cada cliente tendrá, al menos, los siguientes campos:

- Nombre de usuario.

- Estado (Conectado / Desconectado).
- IP (en caso de estar conectado).
- Puerto (en caso de estar conectado).
- Lista de mensajes pendientes.
- Identificador del último mensaje recibido.

Una vez registrado un cliente, el servidor mostrará el siguiente mensaje por consola en caso de éxito:

```
s> REGISTER <userName> OK
```

En caso de que se haya producido un error en el registro se mostrará:

```
s> REGISTER <userName> FAIL
```

5.3. Baja de un cliente

Cuando un cliente quiera darse de baja del servicio de mensajería debe enviar el mensaje correspondiente indicando en él el nombre de usuario que se quiere borrar. Cuando el servidor recibe el mensaje hará lo siguiente:

- Verificar que el usuario está registrado.
- Si el usuario existe, se borra su entrada de la lista y se envía un 0 al cliente.
- Si no existe, se envía una notificación de error al cliente (código con valor 1).

Cuando se realice con éxito el borrado del usuario se mostrará por consola el siguiente mensaje:

```
s> UNREGISTER <userName> OK
```

En caso de fallo se mostrará:

```
s> UNREGISTER <userName> FAIL
```

Cuando un usuario se da de baja del sistema, se borrarán todos los mensajes (en caso de no estar conectado) que todavía no se le han entregado.

5.4. Conexión de un cliente

Cuando un cliente quiere notificar al servidor que está disponible para recibir mensajes por parte de otros usuarios, debe indicar su puerto en un mensaje (la IP se obtendrá a través de la llamada `accept`). Cuando este mensaje se recibe, el servidor debe realizar lo siguiente:

- Buscar el nombre de usuario indicado entre todos los usuarios registrados.
- Si el usuario existe y su estado es "Desconectado":
  - Se rellena el campo IP y puerto del usuario.
  - Se cambia su estado a "Conectado".
  - Se devuelve el código de la operación (0).

Si una vez conectado, existen mensajes pendientes de enviar para este usuario, se enviarán todos los mensajes al usuario uno a uno.

Si el usuario no existe, se devuelve un 1, si ya está conectado un 2, y en cualquier otro caso un 3.

Cuando la operación de conexión finaliza con éxito en el servidor se debe mostrar el siguiente mensaje por la consola:

```
s> CONNECT <userName> OK
```

En caso de fallo se mostrará:

```
s> CONNECT <userName> FAIL
```

Si existen mensajes pendientes para el usuario que se ha conectado se mostrará el siguiente mensaje por cada uno de ellos que se envíe:

```
s> SEND MESSAGE <id> FROM <userNameS> TO <userNameR>
```

Siendo `userNameS` el usuario que envió el mensaje originalmente, `userNameR` el usuario que destinatario del mensaje, y `id` el identificador asociado al mensaje que se envía.

5.5. Desconexión de un cliente

Cuando un cliente quiere dejar de recibir mensajes del servicio debe enviar el mensaje correspondiente indicando el nombre de usuario. Cuando el servidor reciba este mensaje realizará lo siguiente:

- Buscar el nombre de usuario indicado entre los usuarios registrados.
- Si el usuario existe y su estado es "Conectado":
  - Borra los campos IP y puerto del usuario.
  - Se cambia su estado a"Desconectado".
- Si no existe, se envía una notificación de error al cliente.

Cuando la operación finaliza con éxito, se debe mostrar por consola lo siguiente:

```
s> DISCONNECT <userName> OK
```

En caso de error, se mostrará:

```
s> DISCONNECT <userName> FAIL
```

5.6. Envío de un mensaje

Cuando un cliente quiere enviar un mensaje a otro cliente registrado deberá enviar el mensaje correspondiente al servidor indicando el usuario de destino, su nombre y el mensaje. Una vez recibido el mensaje en el servidor, éste realizará lo siguiente:

- Buscar el nombre de ambos usuarios entre los usuarios registrados.
- Si uno de los dos usuarios no existe se envía un mensaje de error al cliente (ver sección 5).
- Se almacena en la lista de mensajes pendientes del usuario destino el mensaje junto con el usuario que lo envía y el identificador asignado.
- Se devuelve un mensaje al remitente con el identificador de mensaje asignado (código 0, cuando todo ha ido bien).

Una vez realizadas estas acciones, si el usuario destino existe y su estado es "Conectado":

- Se envía el mensaje a la IP:puerto indicado en la entrada del usuario.
- Se envía al cliente remitente del mensaje un mensaje indicando que el mensaje con el identificador correspondiente se ha enviado.

Una vez finalizado el envío se muestra por la consola del servidor el siguiente mensaje:

```
s> SEND MESSAGE <id> FROM <userNameS> TO <userNameR>
```

Si el usuario destino existe y su estado es "Desconectado", no realizará ninguna acción. Los mensajes se enviará en el momento en el que el proceso destinatario del mensaje se conecte. En este caso se mostrará por pantalla:

```
s>  MESSAGE <id> FROM <userNameS> TO <userNameR> STORED
```

Los mensajes almacenados se enviarán posteriormente (uno a uno) cuando el cliente destinatario se conecte al sistema.

6. Protocolo de comunicación

En este apartado se especificarán los mensajes que se intercambiarán el servidor y los clientes. Estos mensajes no se pueden modificar y se deben usar tal y como se describen y en el orden en el que se describen. En todo el protocolo se establece una conexión por cada operación.

IMPORTANTE: todos los campos enviados se codificarán como cadenas de caracteres. Se recuerda que las cadenas finalizan con el código ASCII `'\0'`.

6.1. Registro

Cuando un cliente quiere registrarse realiza las siguientes operaciones:

1. Se conecta al servidor, de acuerdo a la IP y puerto pasado en la línea de mandatos al programa.
2. Se envía la cadena "REGISTER" indicando la operación.
3. Se envía una cadena de caracteres con el nombre del usuario que se quiere registrar y que identifica al usuario.
4. Recibe del servidor un byte que codifica el resultado de la operación: 0 en caso de éxito, 1 si el usuario ya está registrado previamente, 2 en cualquier otro caso.
5. Cierra la conexión.

Recuerde que todas las cadenas finalizan con el código ASCII `'\0'`.

6.2. Baja

Cuando un cliente quiere darse de baja envía se realizan las siguientes operaciones::

1. Se conecta al servidor, de acuerdo a la IP y puerto pasado en la línea de mandatos al programa.
2. Se envía la cadena "UNREGISTER" indicando la operación.

3. Se envía una cadena con el nombre del usuario que se quiere dar de baja.
4. Recibe del servidor un byte que codifica el resultado de la operación: 0 en caso de éxito, 1 si el usuario no existe, 2 en cualquier otro caso.
5. Cierra la conexión.

6.3. Conexión

Cuando un cliente se quiere conectar al servicio debe realizar las siguientes operaciones::

1. Se conecta al servidor, de acuerdo a la IP y puerto pasado en la línea de mandatos al programa.
2. Se envía la cadena "CONNECT" indicando la operación.
3. Se envía una cadena con el nombre del usuario.
4. Se envía una cadena de caracteres que codifica el número de puerto de escucha del cliente. Así, para el puerto 456, esta cadena será "456".
5. Recibe del servidor un byte que codifica el resultado de la operación: 0 en caso de éxito, 1 si el usuario no existe, 2 si el usuario ya está conectado y 3 en cualquier otro caso.
6. Cierra la conexión.

6.4. Desconexión

Cuando el cliente quiera dejar de recibir los mensajes deberá realizar las siguientes acciones:

1. Se conecta al servidor, de acuerdo a la IP y puerto pasado en la línea de mandatos al programa.
2. Se envía la cadena "DISCONNECT" indicando la operación.
3. Se envía una cadena con el nombre del usuario que se desea desconectar.
4. Recibe del servidor un byte que codifica el resultado de la operación: 0 en caso de éxito, 1 si el usuario no existe, 2 si el usuario no está conectado y 3 en cualquier otro caso.
5. Cierra la conexión.

Tenga en cuenta que un usuario solo se puede desconectar si la operación se envía desde la IP desde la que se registró.

6.5. Envío de un mensaje cliente-servidor

Cuando el cliente quiere enviarle a otro usuario un mensaje realizará las siguientes acciones:

1. Se conecta al servidor, de acuerdo a la IP y puerto pasado en la línea de mandatos al programa.
2. Se envía la cadena "SEND" indicando la operación.
3. Se envía una cadena con el nombre que identifica al usuario que envía el mensaje.

4. Se envía una cadena con el nombre que identifica al usuario destinatario del mensaje mensaje.

5. Se envía una cadena en la que se codifica el mensaje a enviar (como mucho 256 caracteres incluido el código '0', es decir la cadena tendrá como mucho una longitud de 255 caracteres).

6. Recibe del servidor un byte que codifica el resultado de la operación: 0 en caso de éxito. En este caso recibirá a continuación una cadena de caracteres que codificará el identificador numérico asignado al mensaje, "132" para el mensaje con número 132. Si no se ha realizado la operación con éxito se recibe 1 si el usuario no existe y 2 en cualquier otro caso. En estos dos casos no se recibirá ningún identificador.

7. Cierra la conexión.

6.6. Envío de un mensaje servidor-cliente

Cuando el servidor quiere enviarle a un usuario registrado y conectado un mensaje de otro usuario realizará las siguientes acciones (por cada mensaje a enviar):

1. Se conecta al thread de escucha del cliente (de acuerdo a la IP y puerto almacenado para ese cliente).

2. Se envía la cadena "SEND_MESSAGE" indicando la operación.

3. Se envía una cadena con el nombre que identifica al usuario que envía el mensaje.

4. Se envía una cadena codificando en ella el identificador asociado al mensaje.

5. Se envía una cadena con el mensaje (todos los mensajes tendrán como mucho 256 bytes incluido el código '0', este tamaño lo controlará el cliente).

6. Cierra la conexión.

Si se produce algún error durante esta operación, el mensaje se considerará no entregado y se seguirá almacenando en el servidor como pendiente de entrega, hasta que se pueda entregar. Si se produce algún error durante la conexión a un cliente, el servidor asumirá que el cliente se ha desconectado y lo marcará como desconectado.

Cuando el servidor notifica a un cliente remitente de un mensaje que el mensaje se ha entregado con éxito a otro realiza las siguientes acciones:

1. Se conecta al thread de escucha del cliente (de acuerdo a la IP y puerto almacenado para ese cliente).

2. Se envía la cadena "SEND_MESS_ACK" indicando la operación.

3. Se envía una cadena codificando en ella el identificador asociado al mensaje que se ha entregado.

4. Cierra la conexión.