Distributed Systems

Bachelor in Informatics Engineering

Universidad Carlos III de Madrid

Assignment

Design and implementation of a communication system (Part I)

Felix Garcia Carballeira
Francisco Javier Garcia Blas

March 17, 2021

Contents

1 Objective **2**

I Message Notification Service **2**

2 Introduction **3**
2.1 Development 3

3 Client implementation **4**
3.1 Client usage 4
3.2 Registration on the system 4
3.3 Remove a registration on the system 5
3.4 Perform a connection on the system 5
3.4.1 Message reception 6
3.5 Disconnection of the system 7
3.6 Message transmission 7

4 Server implementation **8**
4.1 Server usage 9
4.2 Registration of clients 9
4.3 Clients removal 10
4.4 Connection of a client 10
4.5 Disconnection of a client 11
4.6 Message transmission 11

5 Communication protocol **12**
5.1 Register 12
5.2 Unregister 13
5.3 Connect 13
5.4 Disconnect 13
5.5 Message passing from client to server 14
5.6 Message passing from server to client 14

1 Objective

The objective of this assignment is to get the main concepts related to the design and development of a distributed application that uses different technologies (Sockets, RPC, SOAP Web Services, and REST).

Functionality

The objective of this assignment is to develop a message notification service between users, in a similar way to what happens with the **WhatsApp** application. We can send text messages of a maximum size of 256 bytes (including code 0 that indicates end of string, that is, at most the string stored in the message will have a maximum length of 255 characters) and optionally, we can also send attachments of any size. The diagram of the application is shown in Figure 1.

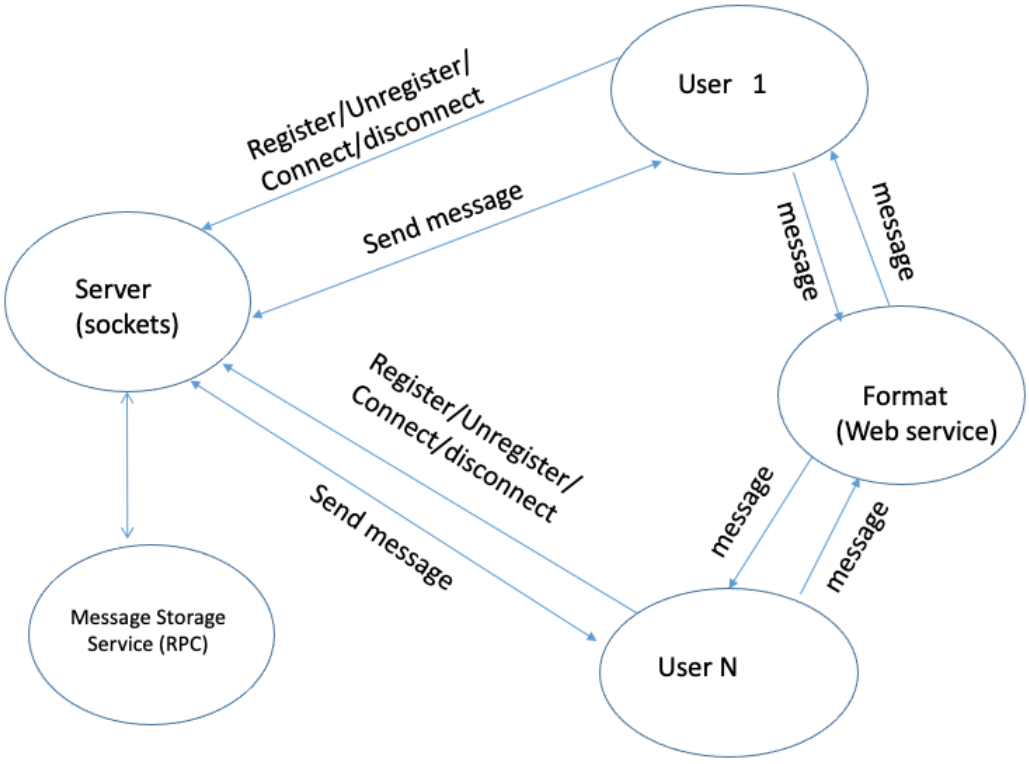

Figure 1: Full vision of the proposed distributed system.

The implementation of this assignment is done in a incremental way. In a first part, you should develop the functionality shown in Figure 3 . In this first part, it is not be necessary to implement the attachment feature. This functionality will be included later on.

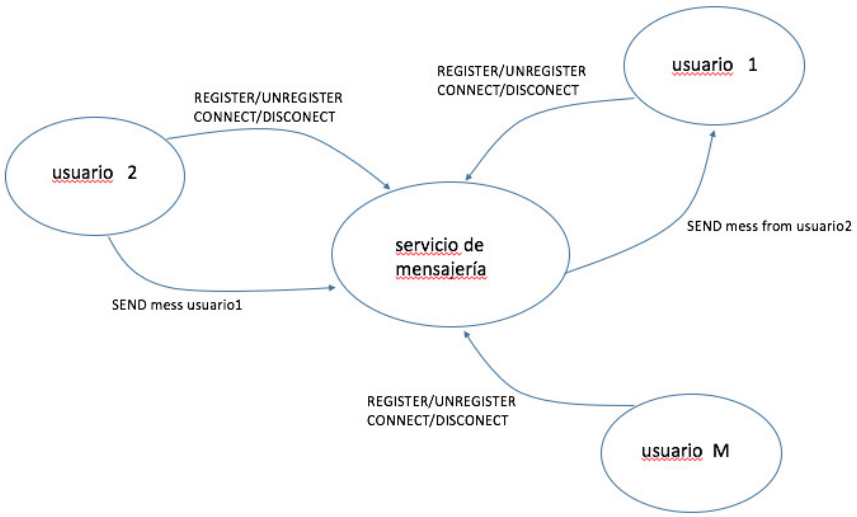

Figure 2: Notification service to be developed in the first part.

Part I
Message Notification Service

2 Introduction

We want to implement a publication/subscription service as described in Figure 3. The application to be developed consists of three separate modules or components.

The student should design, implement, and test, using the C language and on a UNIX/Linux Docker system, a server that manages the sending and receiving of messages between the different clients registered in the platform. On the other hand, you should design, implement, and test, using the Java language, a client that communicates with the server and permits to send messages to other clients, as well as the reception of messages sent by other clients.

2.1 Development

It is intended the implementation of two applications:

- A multi-threaded concurrent server that provides the communication service between the different clients registered in the system, manages the connections of the same, and stores the messages sent to a client not directly connected to the system.
- A multi-threaded client that communicates with the server and is able to send and receive messages. One of the threads will be used to send messages to the server and the other to receive them.

3 Client implementation

The client will execute a command interpreter to communicate with the server following the same protocol as this one. The communication protocol is detailed in Section 5.

3.1 Client usage

This interpreter is provided as support material.

Uncompress the file `assignment1.tgz`:

```
$tar xvf assignment1.tgz
```

Inside this directory you will find a *client.java* file . To compile this file it is recommended to use a Cmake project.

You can then run the client program as follows:

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

Where **server** and **port** represent the IP address and port of the server. The name **server** can be both the name and IP address of the server. The command interpreter will display:

```
c>
```

And will be waiting for the user to enter commands. To end this interpreter, the client must write **QUIT**.

```
c> QUIT
```

Throughout the execution of the client can be obtained errors due to the network (server down, failure in the network, etc.), all these errors must be taken into account. Therefore, it is recommended to make a correct treatment of the errors returned by the system calls and the socket library used.

The provided client includes all the code to read the commands entered by the user and described below.

3.2 Registration on the system

When a client wants to register as a new user, the following command must be executed in the client console:

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

This operation can return three results (whose values are described in detail in the section intended to describe the communication protocol): 0 if the operation was successful, 1 if there is already a registered user with the same name, and 2 in any other case . If the operation is successful, the client will receive a message with code 0 and will display the following message by console:

```
c> REGISTER OK
```

If the user is already registered, it will be displayed:

```
c> USERNAME IN USE
```

In this case the server will not perform the registration of this user.

In case the registration operation can not be performed, either because the server is down or because code 2 is returned, the following message will be displayed on the console:

```
c> REGISTER FAIL
```

3.3 Remove a registration on the system

When a client wants to unsubscribe from the messaging service, she must execute the following command in the console:

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

This operation can return three results: 0 if the operation is successful, 1 if the user does not exist and 2 in any other case. If the user who wants to unsubscribe from the service does not exist, the server will return a 1 (described in the section intended to describe the communication protocol) and the following message will be displayed by the client console:

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

If the unsubscribe operation is successful and the user is deleted on the server, the server will return a 0 and the client will display through the console:

```
c> UNREGISTER OK
```

In case this operation can not be performed, either because the server is down or because it returns the code 2, the following message will be displayed in the console:

```
c> UNREGISTER FAIL
```

3.4 Perform a connection on the system

Once a client is registered on the message system, the message system can connect and disconnect from the service as many times as desired. To connect, you must send (using the protocol described in Section 5) to the server your IP address and port number, so that it can send you the messages of other users.

To do this the client will enter by console:

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

Internally, the client will search for a valid free port. Once the port is obtained, and before sending the message to the server, the client must create a thread that will be in charge of listening (on the IP and port selected) and attend to the messages sent by other users from the server.

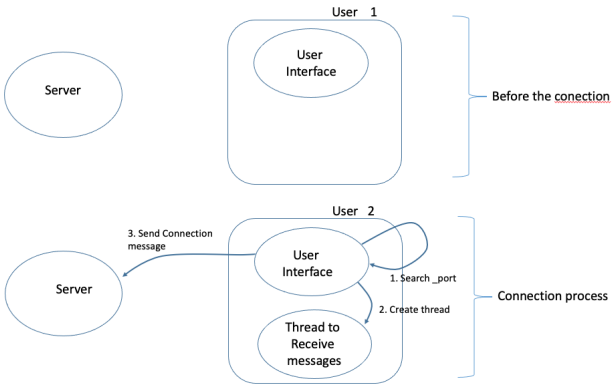

Figure 3: Structure of a client process connected to the messaging service.

Once the connection is established on the system, the server will return a byte that will encode the result of the operation: 0 on success, 1 if the user does not exist, 2 if the user is already connected, and 3 in any other case.

If everything went well, it will be shown by console:

```
c> CONNECT  OK
```

In case of code 1 (user is not registered in the system), the client will display the following error per console:

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

In case the client was already connected in the system (code 2), the client will show through the console:

```
c> USER ALREADY  CONNECTED
```

In case the connection operation can not be performed, either because the server is down, communications fail or code 3 is returned, the following message will be displayed on the console:

```
c> CONNECT FAIL
```

3.4.1 Message reception

Each time the client receives a message through the dedicated thread created for it, it should display by console the following message:

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

As will be seen in the communication protocol section, the reception of the message will include the sender and the message and an identifier (integer) that identifies it.

3.5 Disconnection of the system

When a client does not want to continue receiving messages from the server (but does not want to unsubscribe from the service), it must execute the following command by console:

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

Internally, the client must stop the execution of the thread created in the CONNECT operation. The server can return 3 values: 0 if it was successful, 1 if the user does not exist, 2 if the user is not connected, and 3 in case of error.

If everything went correctly, the server will return a 0 and the client will display the following message per console:

```
c> DISCONNECT OK
```

If the user does not exist, the following message will be displayed per console:

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

If the user exists but did not previously connect, the following message will be displayed per console:

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

In case the operation can not be performed with the server, because the server is down, there is an error in the communications or the server returns a 3, the following message will be displayed by the console:

```
c> DISCONNECT FAIL
```

In case of an error in the disconnection process, the client will also stop the execution of the thread created in the CONNECT operation, acting as if the disconnection had been made.

3.6 Message transmission

Once the client is registered and connected to the system, the client can send messages to other registered users. To do this, execute the following command from the console:

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

For implementing this functionality, the server will associate each message sent by a user with an integer as an identifier and will always keep track of the last identifier assigned to a user's message. When a user is registered for the first time in the system, this identifier is set to 0, so the first message is identified as 1, the second as 2, and so on. When the maximum number of possible identifiers is reached, the new identifier to be assigned will return to 1, and proceed in a similar way. The identifier must be stored in a variable of type `unsigned int`, when the maximum number representable in a variable of this type is reached and it is added to 1, the variable will return to value 0 and the process will be continued, so that the next identifier will return to 1.

When a message is sent to the server, it returns a byte with three possible values (described in detail in Section 5): a 0 in case of success, 1 if the user does not exist and 2 in any other case. On success (code 0), it will also return the identifier associated with the sent message (an integer) and the following message will be displayed by console:

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

In case a message is sent to an unregistered user, the server will indicate the error (code 1) and will display by console:

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

In case of an error (server crash, communications error, error due to message storage problems or a type 2 error is returned), it will be displayed by console:

```
c> SEND FAIL
```

Once the server stores a message for a user and has responded with the corresponding code to the sending user, if the user is connected then he will send the message. In case of successful sending, the server will send the sender of the message confirmation that the message with the assigned identifier has been sent to the user successfully (described in detail in Section 5). Each time a sender of a message receives a message from the server to another process, it will display by console:

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

Indicating that the message with identifier `id` has been successfully delivered.

In case the user is not connected, the server will store the message. Subsequently when the client client is connected the server will be responsible for sending all pending messages (one by one). Each time a message is successfully sent to a user, the sender of the message is notified, which will display:

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

As will be seen later, whenever the server successfully sends a message to a user, it discards the message by deleting it from the server. That is, the server only stores the messages pending delivery, every time a message is successfully delivered is deleted from the server.

4 Server implementation

The purpose of the server is to provide a communication service between clients. To do this, clients must register with a specific name in the system and then connect, indicating their IP and port. The server must keep a list of all registered clients, their name, status and address, as well as a list of messages pending delivery to each client. In addition it will be responsible for associating an identifier to each message received from a client.

The server must be able to handle multiple connections simultaneously (it must be concurrent) by using POSIX threads. The server will use connection-oriented sockets (TCP).

4.1 Server usage

It will be executed as follows:

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

When starting the server the following message will be displayed:

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

Before receiving commands from clients, it will show:

```
s>
```

The program will terminate upon receipt of a *SIGINT* signal (Ctrl + c).

4.2 Registration of clients

When a client wants to register, it will send the corresponding message indicating the username. When this message is received, the server should do the following:

- Verify that there is no other registered user with the same username.
- If the user does not exist, the information is stored with the user's name and the code 0 is sent to the client. The value associated with the message identifier is set to 0.
- If it exists, a notification is sent to the customer indicating it.

The information associated with each client will have at least the following fields:

- Username.
- Status (On / Off).
- IP (if connected).
- Port (if connected).
- List of pending messages.
- Identifier of the last message received.

Once a client is registered, the server will display the following message per console on success:

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

In case of an error in the registry, it will be displayed:

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

4.3 Clients removal

When a client wants to unsubscribe from the messaging service, she must send the corresponding message, indicating the user name to be deleted. When the server receives the message it will do the following:

- Verify that the user is registered.
- If the user exists, his entry is deleted from the list and a 0 is sent to the client.
- If it does not exist, an error notification is sent to the client (code with a value of 1).

When the user is successfully deleted, the following message will be displayed by console:

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

In case of failure it will be shown:

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

When a user logs out of the system, all messages (if not logged in) that have not been delivered will be deleted.

4.4 Connection of a client

When a client wants to notify the server that it is available to receive messages from other users, it must indicate its port in a message (the IP will be obtained through the `accept` call). When this message is received, the server must do the following:

- Find the indicated user name among all registered users.
- If the user exists and their status is "Disconnected":
  - The user's IP and port field is filled in.
  - The status is changed to " Connected ".
  - The operation code (0) is returned.

If once connected, there are messages pending to send for this user, all messages will be sent to the user one by one.

If the user does not exist, a 1 is returned, if a 2 is already connected, and in any other case a 3.

When the connection operation ends successfully on the server, the following message must be displayed by the console:

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

In case of failure it will be shown::

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

If there are pending messages for the connected user, the following message will be displayed for each one that is sent:

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

Being `userNameS` is the user who originally sent the message, `userNameR` the user who is the recipient of the message, and `id` is the identifier associated with the message being sent.

4.5 Disconnection of a client

When a client wants to stop receiving messages from the service, she must send the corresponding message indicating the user name. When the server receives this message it will do the following:

- Find the indicated user name among registered users.
- If the user exists and his status is "Connected":
  - Deletes the user's IP and port fields.
  - The status is changed to "Off".
- If it does not exist, an error notification is sent to the client.

When the operation is successful, the following should be displayed by console:

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

In case of error, it will be shown:

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

4.6 Message transmission

When a client wants to send a message to another registered client, it must send the corresponding message to the server indicating the destination user, its name and the message. Once the message is received on the server, the server will do the following:

- Find the names of both users among registered users.
- If one of the two users does not exist an error message is sent to the client (see Section 5).
- The message along with the sending user and the assigned identifier are stored in the destination user's pending message list.

- A message is returned to the sender with the assigned message identifier (code 0, when all has gone well).

Once these actions are performed, if the destination user exists and their status is "Connected":

- The message is sent to the IP: port indicated in the user input.
- The message is sent indicating the corresponding identifier.

Once the sending is finished, the following message is displayed by the server console:

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

If the destination user exists and its status is "Disconnected", it will not take any action. The messages will be sent the moment the message's destination process connects. In this case, the screen will show:

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

Stored messages will be sent later (one by one) when the recipient client connects to the system.

5 Communication protocol

This section specifies the messages to be exchanged between the server and clients. These messages can not be modified and should be used as described and in the order in which they are described. Throughout the protocol a connection is established for each operation.

IMPORTANT: All fields sent will be encoded as strings. Remember that the strings end with the ASCII `'\0'` code.

5.1 Register

When a client wants to register, she performs the following operations:

1. Connects to the server, according to the IP and port passed in the command line to the program.
2. The string "REGISTER" is sent indicating the operation.
3. A string of characters is sent with the name of the user to be registered.
4. It receives from the server a byte that encodes the result of the operation: 0 in case of success, 1 if the user is previously registered, 2 in any other case.
5. Close the connection.

Remember that all strings end with the ASCII `'\0'` code.

5.2 Unregister

When a client wants to unsubscribe send the following operations are performed:

1. Connects to the server, according to the IP and port passed in the command line to the program.
2. The string "UNREGISTER" is sent indicating the operation.
3. A string is sent with the name of the user who wants to unsubscribe.
4. It receives from the server a byte that encodes the result of the operation: 0 in case of success, 1 if the user does not exist, 2 in any other case.
5. Close the connection.

5.3 Connect

When a client wants to connect to the service, it must perform the following operations:

1. Connects to the server, according to the IP and port passed in the command line to the program.
2. The string "CONNECT" is sent indicating the operation.
3. A string is sent with the name of the user.
4. We send a string of characters encoding the client's listening port number. Thus, for port 456, this string will be "456".
5. It receives a byte from the server that encodes the result of the operation: 0 on success, 1 if the user does not exist, 2 if the user is already connected and 3 in any other case
6. Close the connection.

5.4 Disconnect

When the client wants to stop receiving the messages, the following actions must be taken:

1. Connects to the server, according to the IP and port passed in the command line to the program.
2. The string "DISCONNECT" is sent indicating the operation.
3. A string is sent with the name of the user to be disconnected.

4. It receives from the server a byte that encodes the result of the operation: 0 on success, 1 if the user does not exist, 2 if the user is not connected and 3 in any other case.

5. Close the connection.

Note that a user can only disconnect if the operation is sent from the IP from which it was registered.

5.5 Message passing from client to server

When the client wants to send another user a message will perform the following actions:

1. Connects to the server, according to the IP and port passed in the command line to the program.

2. The string "SEND" is sent indicating the operation.

3. A string is sent with the name that identifies the user sending the message.

4. A string is sent with the name that identifies the recipient user of the message message.

5. A string is sent in which the message to be sent is encoded (at most 256 characters including the code '0', i.e. the string will have a length of 255 characters at most).

6. It receives from the server a byte that encodes the result of the operation: 0 on success. In this case, you will then receive a character string that will encode the numeric identifier assigned to the message, "132" for the message with number 132. If the operation has not been successfully performed, 1 is received if the user does not exist and 2 In any other case. In these two cases no identifier will be received.

7. Close the connection.

5.6 Message passing from server to client

When the server wants to send a message to a connected and registered user from another user, the server will perform the following actions (for each message):

1. Connects to the client listening thread (according to the IP and port stored for that client).

2. The string "SEND_MESSAGE" is sent indicating the operation.

3. A string is sent with the name that identifies the user sending the message.

4. A string is sent by encoding the identifier associated with the message.
5. A string is sent with the message (all messages will have at most 256 bytes including code '0', this size will be controlled by the client).
6. Close the connection.

If an error occurs during this operation, the message will be considered undeliverable and will be stored on the server as pending delivery until it can be delivered. If an error occurs while connecting to a client, the server will assume that the client has disconnected and the server will mark it as disconnected.

When the server notifies a sender of a message that the message has been successfully delivered to another, it performs the following actions:

1. Connects to the client listening thread (according to the IP and port stored for that client).
2. The string "SEND_MESS_ACK" is sent indicating the operation.
3. A string is sent by encoding the identifier associated with the message that has been delivered.
4. Close the connection.