Distributed Systems

Grado en Ingeniería Informática

Project (part 1)

Design and implementation of a peer-to-peer system

Félix García Carballeira
Francisco Javier García Blas

3 de marzo de 2020

Índice

1. Objective

The overall objective of the whole assignment is to develop a peer-to-peer system for distributing files among clients. The assignment will be done in an incremental way through three parts.

The objective of this first part of the assignment is that the student gets to know the main concepts related to the communication of processes using TCP sockets. For this purpose, it is proposed to implement a file distribution application between clients, as described in the following sections.

2. Description of the functionality of the first part

The idea is to develop a peer-to-peer service like the one shown in the Figure 1. The application consists of two independent programs:

- *Client*. This is a program that is executed by the system's clients. This program allows to publish content and it provides files to other users.
- *Server*. It is the server program that coordinates the operation of the application.

Users can register, unsubscribe, connect, and disconnect from the service (these will be messages sent from the clients to the server). Each user connected will be able to publish file names on the server. He can also delete these names from the server. The server will maintain for each client on the system (whether connected or not) the list of file names it has made public.

A logged-in user can request the server for the list of system users and can ask for the list of files published by a particular user. When a user wants to download a file stored on another client, he will ask that client directly. In other words, the server does not store the files shared by the users. The files are held by each user on his machine and he can send them to other clients that request them.

3. First part

The student must design, implement, and test, using the C language and on a UNIX/-Linux operating system, a server that manages the system's functionality and, on the other hand, must design, implement and test, using the Java language and on a UNIX/Linux operating system, the clients'code.

Next, we describe characteristics of the system. In this part of the document, the protocol to be followed between the server and the client will be described. This protocol will allow any client that follows it to communicate with the implemented server. This makes that, different students can test their clients with the servers developed by others.

For the storage of the users and the names of files published by each user, in this first part it will be possible to use lists in memory like the ones used in the first evaluable exercise. You can also use disk storage. Note that later, this information will have to be stored persistently using services that use RPC (remote procedure calls). Each project lab group can choose the desired layout for storing this information.

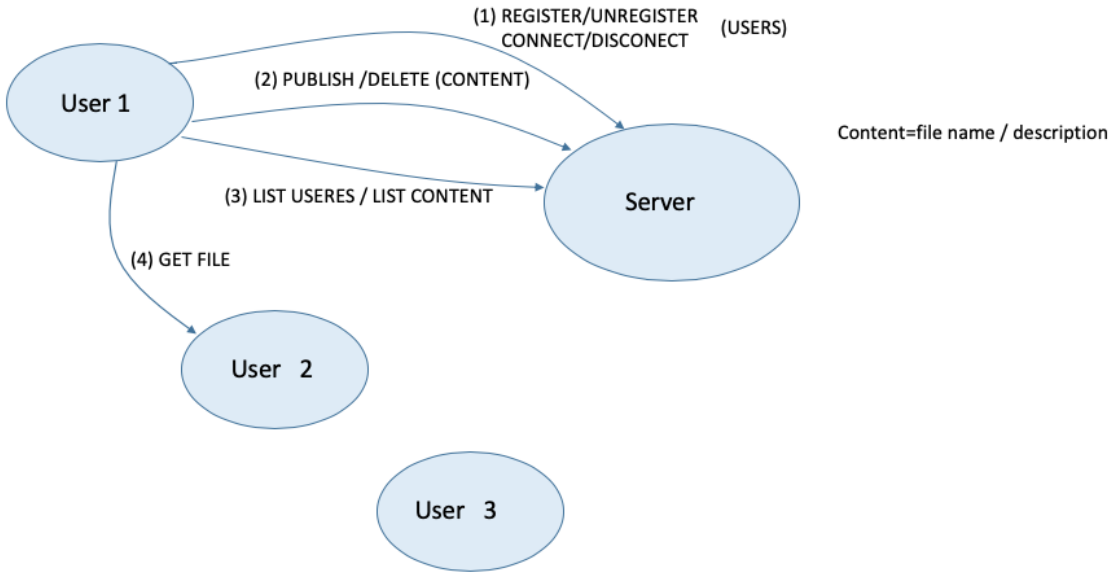

Figura 1: Aplication model.

3.1. Service development

The objective is to design and develop the following two programs:

- A multi-threaded concurrent server (written in C) that provides server functionality.
- A multi-threaded client (written in Java) that offers the functionality that users of the peer-to-peer system execute.

4. Client development and protocol

The client executes a command interpreter to communicate with the server following the protocol described in the following sections.

4.1. Client use

This interpreter is provided as support material. Unzip the `ssdd_pract_material.tar`:

```
$tar xvf ssdd_pract_material.tar
```

Inside this directory, you will find the *client.java* file and all the necessary code to compile it. To compile this file and get its executable class, run

```
$ javac   client.java
```

You can then run the client program as follows:

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

Where **server** and **port** represent the IP address and port where the server is running. The name **server** can be either the name (domain-dot) or the IP address (decimal-dot) of the server. The command prompt will display:

```
c>
```

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

```
c> QUIT
```

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

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

If a user is connected and QUIT is entered, the client must first disconnect from the system.

4.2. Register of a client in the system

When a client wants to register as a user of the service he must execute the following command in the client's console:

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

When a customer wants to register in the system he will perform the following operations:

1. It connects to the server, according to the IP and port passed on the command line to the client program.
2. The string "REGISTER" is sent to the server indicating the operation. The string ends with the ASCII code `'0'`.
3. A string of characters is sent with the name of the user you want to register and which identifies the user.
4. It receives from the server a byte which encodes the result of the operation: 0 in case of success, 1 if the user is already registered, 2 in any other case.
5. Closes the connection.

Remember that all the character strings sent end with the ASCII code `'0'`. The maximum size of a user name is 256 bytes.

This service, once executed on the server, can return three results (0,1 or 2). If the operation is successful, the client will receive from the server a message with code 0 and will show the following message on the 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 make any record.

In case the registration operation cannot be performed, either because the server is down or because the code 2 is returned due to any other error, the following message will be shown in the console:

```
c> REGISTER FAIL
```

4.3. Unregister from the system

When a client wants to unsubscribe from the service he should execute the following command in the console:

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

To do so, it will carry out the following operations:

1. It connects to the server, according to the IP and port passed on the command line to the program.
2. The string "UNREGISTER" is sent to the server indicating the operation.
3. A string is then sent with the name of the user to be deleted.
4. It receives from the server a byte encoding the result of the operation: 0 if successful, 1 if the user does not exist, 2 in any other case.
5. It closes the connection.

This service, once executed on the server, can return three results (0, 1, 2). If the unregister operation is successful and the user is deleted from the server, the server will return a 0 and the client will show through the console:

```
c> UNREGISTER OK
```

If the user you want to unsubscribe from the service does not exist, the following message will be shown by the client's console:

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

In case this operation cannot be done, either because the server is down and the connection with it cannot be established or because it returns the code 2, the following message will be shown in the console:

```
c> UNREGISTER FAIL
```

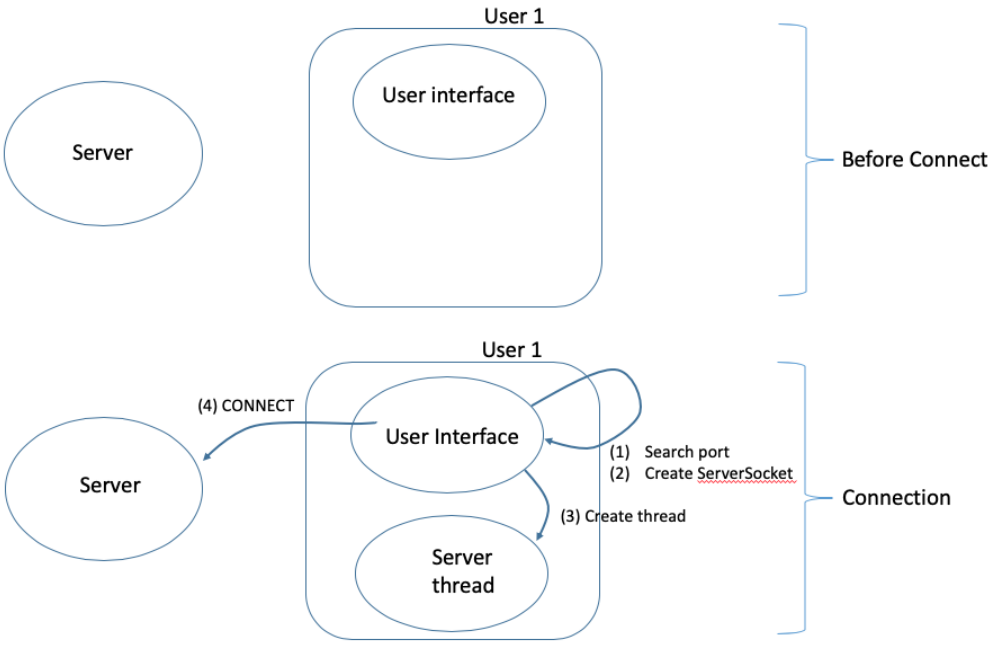

Figura 2: Structure of a client process connected to the service.

4.4. Connection to the system

Once a client is registered in the system, she can log in and out of the service as many times as she wish. To connect, you must send (using the protocol described below) the IP address and port to the server. The structure of a client process connected to the service is shown in Figure 2.

To do this, the customer will enter by console:

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

Internally the client looks for a free port (1). Once the port is obtained, and before sending the message to the server, the client must create a *ServerSocket* (2) and a thread (e) that will be in charge of listening (in the IP and selected port) and attend the download requests of files from other users. Then the client will send (4) the connection request with the following information:

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

6. Closes the connection.

Once the connection has been established in the system, the server will return a byte that will encode the result of the operation: 0 in case of 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 show the following error by console:

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

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

```
c> USER ALREADY  CONNECTED
```

In case the client is already connected in the system (code 2), the client will be shown by the console:

```
c> CONNECT FAIL
```

4.4.1. Publication of content

Each connected client will be able to publish content on the server. This publication will consist of sending a file name and a brief description associated with the file. Only the name of the file is published, not its content. The file is always stored in the client side and the file transfer is done between clients. To publish a content, the client will enter by console:

```
c> PUBLISH  <file name>  <description>
```

Where `<file name>` represents a string with the file name and `<description>` the associated description. The client will do the following operations with the server:

1. It connects to the server, according to the IP and port passed on the command line to the program.
2. The string "PUBLISH" is sent to the server indicating the operation.
3. A string is sent with the name of the file (this string cannot contain blank spaces). The maximum size of the file name will be 256 bytes.
4. A string is sent with the description of the content. This string may contain blank spaces and its maximum size will be 256 bytes.
5. 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 if the user is not connected, 3 the file is already published, and 4 in any other case.
6. Closes the connection.

Once a message has been sent for publication, If everything went well and the code returned by the server is 0:

```
c> PUBLISH  OK
```

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

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

In case the client is NOT connected in the system (code 2), the client will be shown by the console:

```
c> PUBLISH FAIL, USER  NOT CONNECTED
```

In case the file name has already been published by the client, the client will show by the console:

```
c> PUBLISH FAIL, CONTENT ALREDAY PUBLISHED
```

In case the connection operation cannot be made, either because the server is down, there is an error in the communications or the code 4 is returned, the following message will be shown in the console:

```
c> PUBLISH FAIL
```

4.4.2. Delete content

Each connected client will be able to delete content published on the server. The client will send the file name to be deleted from the system. To delete a content the client will enter by console:

```
c> DELETE  <file name>
```

It will then perform the following operations:

1. It connects to the server, according to the IP and port passed on the command line to the program.
2. The string "DELETE" is sent to the server indicating the operation.
3. A string is sent with the name of the user performing the delete operation.
4. A string is sent with the name of the file (this string may not contain blank spaces). The maximum size of the file name is 256 bytes.
5. It receives from the server a byte which encodes the result of the operation: 0 if it is successful, 1 if the user does not exist, 2 if the user is not connected, 3 the file has not been published previously, and 4 in any other case.
6. Closes the connection.

Once a deletion message has been sent, if everything has gone well, it will be displayed on the console:

```
c> DELETE  OK
```

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

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

In case the client is NOT connected in the system (code 2), the client will be shown by the console:

```
c> DELETE FAIL, USER  NOT CONNECTED
```

In case the file name has NOT been previously published by the client, the client will show by the console:

```
c> DELETE FAIL, CONTENT NOT PUBLISHED
```

In case the connection operation cannot be made, either because the server is down, there is an error in the communications or the code 4 is returned, the following message will be shown in the console:

```
c> DELETE FAIL
```

4.4.3. List users

A client connected in the system will be able to know all the users connected in the system. The client will send a request to know the names of the connected users. The server will return the list of connected users with their IP and corresponding port. Only the information of the clients that are connected in the system will be sent. To list the users the client will enter by console:

```
c> LIST_USERS
```

The client will then perform the following operations:

1. It connects to the server, according to the IP and port passed on the command line to the program.
2. The string "LIST_USERS" is sent to the server indicating the operation.
3. A string with the name of the user performing the operation is sent.
4. It receives from the server a byte encoding the result of the operation: 0 in case of success, 1 if the user does not exist, 2 if the user is not connected and 3 in any other case.
5. In case the code is 0, the server will send the following information back to the client
   a) A string encoding the number of users whose information is to be sent. If the string "5" is received, the server will then send the associated information to 5 clients. For each client it will send 3 strings encoding the user name, IP address and port.
6. Close the connection.

If everything went well (code 0), it will be displayed by console:

```
c> LIST_USERS  OK
        USER1  IP1  PORT1
        USER2  IP2  PORT2
        ....
```

indicating the list of connected users and their addresses.

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

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

In case the client is NOT connected in the system (code 2), the client will be shown by the console:

```
c> LIST_USERS FAIL, USER  NOT CONNECTED
```

In case the operation cannot be performed, either because the server is down, there is an error in communications or the code 3 is returned, the following message will be displayed in the console:

```
c> LIST_USERS FAIL
```

4.4.4. List of content

A client connected to the system will be able to know the content published by another user. The client will send a request to the server to know the files published by a certain user. The server will return the list of files published by that user. To list the content published by the user with name `user_name` the client will enter by console:

```
c> LIST_CONTENT <user_name>
```

The client will then perform the following operations:

1. It connects to the server, according to the IP and port passed on the command line to the program.
2. The string "LIST_CONTENT" is sent to the server, indicating the operation.
3. A string with the name of the user performing the operation is sent to the server.
4. A string is sent with the name of the user whose content you want to know (`user_name`).
5. It receives from the server a byte which encodes the result of the operation: 0 in the case of success, 1 if the user who carries out the operation does not exist, 2 if the user who carries out the operation is not connected, 3 if the user whose content you want to know does not exist and 4 in any other case.
6. In case the code is 0 the server will send the following information back to the client:
   a) A string that encodes the number of filenames published. If the string "7" is received, the server will then send the information associated with 7 filenames. For each file, the server will send a string with the name of the file (the maximum size of the file is 256 bytes).

7. Close the connection.

If everything went well (code 0), it will be displayed by console:

```
c> LIST_CONTENT  OK
        file_name_1
        file_name_2
        file_name_3
        ....
```

indicating the list of connected users and their addresses.

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

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

In case the client is NOT connected in the system (code 2), the client will be shown by the console:

```
c> LIST_CONTENT FAIL, USER  NOT CONNECTED
```

In case of code 3 (remote user is not registered in the system), the client will show the following error by console:

```
c> LIST_CONTENT FAIL, REMOTE USER DOES NOT EXIST
```

In case the operation cannot be performed, either because the server is down, there is an error in communications or the code 4 is returned, the following message will be displayed in the console:

```
c> LIST_CONTENT FAIL
```

4.5. Disconnection of the sysgtem

A client can disconnect from the system by entering via the console:

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

Internally the client must stop the execution of the thread created in the CONNECT operation, close its listening port and destroy the thread. The client will perform the following operations:

1. It connects to the server, according to the IP and port passed on the command line to the program.
2. The string "DISCONNECT" is sent to the server, indicating the operation.
3. A string is sent with the name of the user you wish to disconnect.
4. It receives from the server a byte encoding the result of the operation: 0 if it is successful, 1 if the user does not exist, 2 if the user is not connected and 3 in any other case.
5. Closes the connection.

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

```
c> DISCONNECT OK
```

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

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

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

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

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

```
c> DISCONNECT FAIL
```

In the event of an error in the disconnection, the client will stop the execution of the thread created in the CONNECT operation, acting for all purposes as if the disconnection had been made.

This disconnection operation should be done whenever the user enters the QUIT command via the console.

4.6. File transfers

Once the client is registered and connected to the system, he can send messages to other registered users to download the contents of a file. To do so, the following command must be executed through the console:

```
c> GET_FILE <userName> <remote_file_name> <local_file_name>
```

To this end, the client will carry out the following operations:

1. It connects to the client, according to the IP and listening port of that client (whose name `userName` is indicated in the operation and that you will be able to know by means of the operation of list of users .
2. The string "GET_FILE" is sent to the server indicating the operation.
3. A string is sent with the name of the file to be downloaded.
4. It receives from the other client a byte that codes the result of the operation: 0 indicating that the file is going to be transferred, 2 if the file does not exist in the remote client and 2 in any other case. In case of code 0, the remote client will transfer the content of the file closing the connection when the transfer is completed. The local client as it receives the file content will store it in the local file that has been passed to the console.
5. Close the connection.

In case of success (code 0) and once the file has been transferred and stored successfully, the following message will be displayed by console:

```
c>  GET_FILE OK
```

In case the file does not exist (code 1) it will display by console:

```
c> GET_FILE FAIL / FILE NOT EXIST
```

In the event of an error (client down, cannot connect to client, communications error, or any other type of error, or code 2 is returned) it will be displayed by console:

```
c> GET_FILE FAIL
```

In case the transfer could not be completed, the local file will not be created.

5. Server development

The server will be developed to handle all client requests according to the protocol described in the previous sections. The server must be able to manage several connections simultaneously (it must be concurrent) by using multiple threads (multithread). The server must use connection-oriented TCP sockets.

5.1. Server use

It will be executed as follows:

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

When the server is started, the following message will be displayed:

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

Before receiving commands from the clients it will show:

```
s>
```

For each request made by a client, the following information will be displayed on the screen:

```
s> OPERATION FROM USER
```

Where USER is the name of the user performing the operation. The program ends when it receives a SIGINT signal (Ctrl+c).

6. General rules

It is important to analyze the code of support provided with the project since it will be the starting point for the realization of the practice. The skeleton of the server and client program will be provided. These programs should be used as a starting point. All argument and command interpreter processing is already implemented.

1. Practices that do not compile or do not conform to the stated functionality and requirements will get a score of 0.

2. Practices that have warnings or are not commented upon will be penalized. A program that is not commented on will get a grade of 0.

3. The delivery of the practice will be done through the authorized deliverers. It is not allowed to deliver it through e-mail.

4. Special attention will be paid to detect copied functionalities between two practices. In case of detecting a copy, the University regulations will be applied and the students of the groups involved in the copy will have a 0 in the final grade of the course.

5. The practice must work in the computers of the computer classrooms (4.0.F16, 4.0.F18, 2.2.C05 and 2.2.C06) or in guernika.

6. The system must work with all system components running on different machines.

7. The protocol described must be followed carefully.

8. The memory must not exceed 10 pages in length.

7. Documentation to be delivered and delivery time

The practice consists of three parts and will be done incrementally. Only one delivery will be made. The delivery date of the practice will be on **May 3rd**. In the next part of the project, the documentation to be delivered will be indicated.