L01 - UDP

1 Introduction

1.1 Goals

This assignment aims at consolidating the basic concepts required for programming distributed applications directly over the UDP protocol.

You will also gain familiarity with sockets' API, which is adopted by almost all operating systems on the market.

1.2 Method

You can discuss possible solutions with your fellow students. However, the final solution should be your own.

1.3 Follow the program specification

This document is a specification of the application to be developed. As a future engineer, you'll be required to read, understand, and follow specifications, typically elaborated by other people: a solution to a wrong problem is not correct. Get used to follow the specifications of the problems you have to solve. (However, always maintaining a critical attitude to find any inconsistencies or errors.)

2 The application

In this assignment you will write a client-server application to manage a small database of license plates. For any given vehicle, the system should allow to store its plate number and the name of its owner.

Thus, the server must support the following operations:

`register`
to register the association of a plate number to the owner. Returns -1 if the plate number has already been registered; otherwise, returns the number of vehicles in the database.

`lookup`
to get the owner of a given plate number. Returns the owner's name or the string NOT_FOUND if the plate number was never registered.

3 Client-Server Communication Protocol

In a client-server application, the client sends requests to the server. The latter processes the requests, returning the results, if any, to the clients.

In an application based on sockets both the requests and the results are sent in messages. The message sent by the client should include the operation and its parameters, if any. It is the client's responsibility to build the request message. The server will have to extract the operation and the parameters of each message received from a client, and then invoke the function performing the requested operation. Furthermore, if the request elicits a reply message, the server will have to build it , possibly including the results of processing the request. The client in turn will have to interpret the reply message, if any.

The format and meaning of messages exchanged between the client and server, and the rules that govern the exchange of these messages makes the communication protocol between the client and server.

In this work, the server must be able to reply to two distinct requests:

- registration of plate number and its owner;
- query of the owner of given plate number

A possible format for the messages the client will send to the server is:

`‘‘REGISTER <plate number> <owner name>’’`

`‘‘LOOKUP <plate number>’’`

where

`<plate number>`
  is the plate number in the format XX-XX-XX, where X is a digit (0..9) or a letter (A..Z).
`<owner name>`
  is a string with up to 256 characters with the name of the owner of the vehicle.

A possible format for the reply messages by the server to this requests is, respectively:

`<result>`
`<plate number> <owner name>`

where

`<result>`
  is the return (an integer) value of the register command. Return -1 if the command fails; otherwise, returns the number of vehicles in the database. .

Note: To simplify the communication, each message, request or reply, should be a single string.

Question (discuss in class): Note that all messages are plain strings to make the protocol easily readable (e.g, to simplify the debugging phase, if needed). What are the advantages of using plain strings? And what are the drawbacks? Do you think that most application layer protocols, e.g. FTP and HTTP-based protocols use strings? What about lower-level protocols?

4 Communication with DGRAM *Sockets*

In this assignment you should implement two classes – `Client` and `Server`, for the client and server respectively – using the `java.net.DatagramSocket` class.

Server

The server shall be invoked as follows:

> `java Server <port_number>`
> where
>
> > `<port_number>` is the port number on which the server waits for requests.

It must execute in an infinite loop waiting for client requests, processing, and reply to them.

To check if your solution is working, the server should print messages to reveal its activity:

`<oper> <opnd>*`
where:

`<oper>`
is `‘‘register’’` or `‘‘lookup’’`;

`<opnd>*`
is the list of arguments received with the request.

Client

The client must be invoked as follows:

> `java Client <host_name> <port_number> <oper> <opnd>*`
> where
>
> > `<host_name>` is the name of the host running the server;
> > `<port_number>` is the server port;
> > `<oper>` is either `‘‘register’’`or `‘‘lookup’’`
> > `<opnd>*` is the list of arguments
> >
> > > `<plate number>` `<owner name>`, for `register`;
> > > `<plate number>`, for `lookup`.

After submitting a request, the client waits to receive a reply to the request, prints the reply, and then terminates.

The client application should also print messages in the screen to check if it is working as expected, with the following the format:

`<oper> <opnd>*: <result>`
where `<oper>` and `<opnd*>` are the arguments of the request

`<result>`
is the value returned by the server or `‘‘ERROR’’` if an error occurs.

Food for thought: How can you prevent the client from hanging indefinitely in case of failure of the server or of lost messages? Is this a problem for the server?

5 Documentation

- **Sun's tutorial about the Java API for UDP**
- **Java API for UDP communication**
- **User Datagram Protocol**

ADMINISTRAÇÃO DA PÁGINA/UNIDADE