Distributed Systems

Programming with Unicast Datagram Sockets

---

1. Introduction

The main goal of this lab is to reinforce your understanding of the fundamentals for the development of distributed applications using UDP.

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

1.1 Method

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

1.2 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 resolve DNS-like names, i.e. to retrieve the IP address corresponding to a DNS name.

Thus, the server must support the following operations:

register
: to bind a DNS name to an IP address. It returns -1, if the name has already been registered, and the number of bindings in the service, otherwise.

lookup
: to retrieve the IP address previously bound to a DNS name. It returns the IP address in the dotted decimal format or the `NOT_FOUND` string, if the name was not previously registered.

IMP.- Your service is not supposed to use DNS to respond to a `lookup` request. Instead, it should keep a "table" with pairs `(DNS name, IP address)`, which is looked up when processing the `lookup` request. The processing of a `register` request should add a `(DNS name, IP address)` pair to that "table".

3. Client-server application-level 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 from 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 <DNS name> <IP address>’’

‘‘LOOKUP <DNS name>
```

where

<DNS name>
: Is a DNS-like name, e.g. `www.fe.up.pt`

<IP address>
: Is an IPv4 address in dotted decimal format, e.g. `10.227.240.205`

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

```
<result>
<DNS name> <IP address>
```

where

<result>
: is the return value (an integer) of the register command: return -1 if the command fails; otherwise, returns the number of DNS names previously registered in the service.

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?

The application protocol for communication between the client and server can be identical to that used in the TCP version of the service.

4. Unicast Communication with Java's `DatagramSocket`

In this lab you shall implement two classes, **`Client`** and **`Server`**, for the client and server, respectively, using the **`java.net.DatagramSocket`** class.

4.1 Server

The server program shall be invoked as follows:

```
java Server <port number>
```

where

`<port number>`
: is the port number the server shall use to provide the service

The server must loop forever waiting for client requests, processing and replying to them.

To trace the operation of the server, it should print a messages on its terminal each time it processes a client request. The format of this message shall be:

```
Server: <oper> <opnd>*
```

where

`<oper>`
: is operation received on the request

`<opnd>*`
: is the list of operands receive in the request

4.2 Client

The client program shall be invoked as follows:

```
java Client <host> <port> <oper> <opnd>*
```

where

`<host>`

is the DNS name (or the IP address, in the dotted decimal format) where the server is running

`<port>`
: is the port number where the server is providing service

`<oper>`
: is the operation to request from the server, either `"register"` or `"lookup"`

`<opnd>*`
: is the list of operands of that operation
`<DNS name> <IP address>` for `register`
`<DNS name>` for `lookup`

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

The client should print a messages on its terminal to check the operation of the service. The format of this message shall be:

```
Client: <oper> <opnd>* : <result>
```

where

`<oper>`
: is operation requested

`<opnd>*`
: is the list of operands of the request

`<result>`
: is result 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
- RFC768: User Datagram Protocol