PR - 2
ING2 – GSI – Programcion redes
Année 2016–2017

EISTI

UDP Sockets

Here we will use the UDP protocol, that is, the communication will be without connection. There is no connection of any kind and in each transmitted message the address of the target computer must be indicated.

Write the two programs according to the following instructions.

— Include headers: `stdio.h,string.h,stdlib.h,unistd.h,netinet/in.h, arpa/inet.h, sys/types.h y sys/socket.h`

— For each of the programs it is enough with declaring a single socket variable of type int, but we have to declare two structures `sockaddr_in` for the addresses, one for the address of the own socket and another to indicate the remote socket, plus a variable type `socklen_t` for the size of the structures and an array of char that will contain the message.

— Fill in the two addresses `sockaddr_in` with the own and remote data (IP, port and family) in both programs (in the side that will receive really only need to fill in the address itself). We can put any own IP address with `htonl(INADDR_ANY)`. If the remote side is the same computer, we put the remote IP with `inet_addr ("127.0.0.1")`, taking into account that, in this case, the port numbers must be different. Second, get the size of the steering structures. Finally, insert the message to send in its corresponding variable.

— Create a socket in each program with the socket function. Put as arguments, for domain `PF_INET(IPv4)`, to style `SOCK_DGRAM` (no connection), and for protocol you can set 0.

— Assign to the socket the address `sockaddr_in` which it will use with the bind function. Only one socket and one call to the bind function per program is used. One side must use one of the addresses and the other the other one.

— The first of the programs will send the message to the second. The send function is `sendto` and its arguments are the socket number, message, message size, flags (which can be set to 0) and the address structure and size of the target socket (of the second program). The output is the number of bytes sent by the function (display on the screen).

— The second of the programs will receive the message using the function `recvfrom` which has as parameters the socket number, the message, the size of the message (here you have to put the size of the array used as storage since we do not know a priori the size you will have The message that we will receive) and the flags (that we can put 0). The function gives us the structure and size of the address of the source socket (first of the program) whenever we have reserved a space and indicated a size. If we are not interested in this information we can put in both NULL. The output is the number of bytes received by the function. Display the IP address and port of the sender, the number of bytes received and the message.

— Lastly, close each socket with the close function.

Exercises

Maintain the versions of the programs corresponding to the different exercises.

(1) Check what it happens if the `recvfrom` function reads less data than the sendto sent it. Can the remaining data be recovered with a new `recvfrom` statement? □

(2) Other types of data. Modify the programs so that instead of transmitting text strings, an array of float type numbers is transmitted. □

(3) Echo service. Convert programs to an echo server and to an echo client, so that: A) the client transmits to the server what the user type by keyboard, b) the server returns it to the client passed to caps, C) the client shows it on the screen, D) back to step a) Have the server handle multiple clients (sequentially) by inserting a loop:
`while(1) recvfrom(); sendto();` □

Java Sockets

To compile these programs you can use:
`javac ProgramName.java`
To execute them:
`java ProgramName`

IP

(4) Prepare a Java program that shows the name and local address of our computer, as follows:

- The packages needed by the program are `java.io. * and java.net. *`.
- Define a new class with a single main method of type public static void and with for `String argvc []`. Add `throws Exception` if we do not want to worry about managing the exceptions.
- Define an `InetAddress` object to hold the IP addresses.
- Obtain the object that contains the name and IP address of the local computer. Convert text strings to the name and IP address and display them on the screen using the `System.out.println (...)` method.

□

(5) Write a Java program that provides the name and IP address from the computer name given in the command line (next to the name of the executable). □

TCP

(6) As a first exercise, you program a server that sends a text message to the client.

- The packages needed by the program are `java.io. * and java.net. *`.
- Define a new class with a single main method of type public static void and with parameters `String argvc[]`. Add `throws Exception` if you do not want to worry about handling exceptions.
- Define an object of type `String` to contain the messages.
- Define a server type socket and prepare it so that it is attentive to the connections that potential clients can make. As a port choose any one greater than 1024.
- Accept a connection by creating a connection socket at the same time.
- Create an output stream on the server with class `DataOutputStream`.
- Prepare a text string (String) ending in the character ' n'. Send this message with the method `writeBytes(String)`
- Close the output stream and also close the sockets.
- Put the server to run. Try it by typing `telnet 127.0.0.1 port` in another window. The message sent must be displayed.

□

7 As a second exercise, prepare a client that displays on the screen what the server sends to you. The differences are:

- The type of socket that we need is of type client and does not have to prepare it to accept connections. As a computer we can put 127.0.0.1 and the port has to be the same as the server. With this, the connection to the server is requested automatically.
- In the client we will prepare an input stream with `BufferedReader` and `InputStreamReader` because in this case we want to receive.
- Read the string received with the `.readLine()` method. Display it on the screen with the `System.out.println(String)` method.
- Start the server and then run the client from another window. We must receive the message sent to us by the server.

□

8 Modify the programs so that an array of float data can be sent and received. □

9 We can have the server accept multiple client connections by inserting a while loop (true) before the connections. Verify that the server does not finish once you have serviced the first client. □

10 Write a server and an echo client, as follows:

- Returning an echo involves the following steps: A) the client transmits to the server what the user type by keyboard, b) the server returns it to the client passed to caps, C) the client shows it on the screen, D) back to step a)
- We need a new input stream that reads the input from the keyboard with the `BufferedReader` and `InputStreamReader` classes. In this case, the reading is done from the keyboard with the parameter `System.in`.
- Ensure that all strings that are sent end with the ' n' character.
- The method that passes a `String` to caps is `toUpperCase()`.

□

11 You can modify the previous programs so that in the server the port is a parameter entered by the user following the name of the program, while in the client we can choose the port and the computer with which we are going to connect. □

UDP

12 Write programs to transmit a text message from one end to another using datagrams. One of the programs (the sender) will send a message and the other (the receiver) will receive it and display it on the screen.

— The sender needs a datagram type socket with a `port1`, while the receiver uses a socket with a `port2`. The sender will send the datagram packet to `port2`. Both ends will work with address `127.0.0.1`, although the two programs can be generalized so that they can operate in different computers.

— First you have to start the receiver and then run the transmitter. The receiver will display the message on the screen and both programs will exit.

□

13 Convert the previous programs into an echo server and into an echo client. Have the server handle multiple requests with a `while(true)` loop. The server must determine the client computer and port before retransmitting the echo. □