Mandatory exercise 1

thhk@itu.dk, jglr@itu.dk, emja@itu.dk, krbh@itu.dk

August 2020

1

We chose port 7007 as incoming on server and outgoing on client, and port 1337 as outgoing on server and incoming on client.

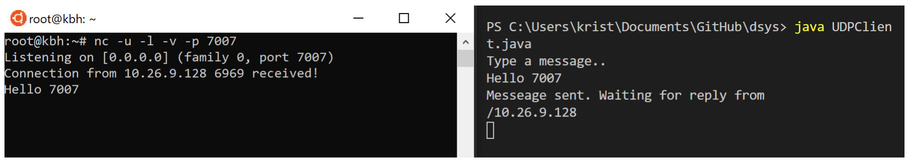
2

Using a network utility tool, like netcat, start to listen on your machine to your selected port.

See 3.

3

In another terminal window, connect to your selected port, and send a message.



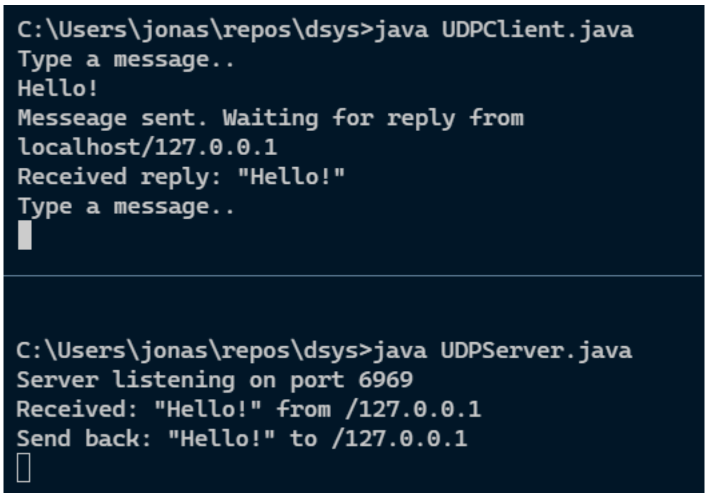
```
root@kbh:~# nc -u -l -v -p 7007
Listening on [0.0.0.0] (family 0, port 7007)
Connection from 10.26.9.128 6969 received!
Hello 7007
```

```
PS C:\Users\krist\Documents\GitHub\dsys> java UDPClien
t.java
Type a message..
Hello 7007
Messeage sent. Waiting for reply from
/10.26.9.128
```

Figure 1: Netcat

4

Write a Java program that given a message sends the same message back (UDP echo server)



```
C:\Users\jonas\repos\dsys>java UDPClient.java
Type a message..
Hello!
Messeage sent. Waiting for reply from
localhost/127.0.0.1
Received reply: "Hello!"
Type a message..
```

```
C:\Users\jonas\repos\dsys>java UDPServer.java
Server listening on port 6969
Received: "Hello!" from /127.0.0.1
Send back: "Hello!" to /127.0.0.1
```

Figure 2: Demonstration of the program

5

Check in the terminal, that you have received the message string from your Java program.

See 4.

6

What do you need to change in your program, in order to send the message to another group members computer?

Instead of localhost use local IP addresses.

7

Discuss: what can possibly go wrong with your program - how does Java express the possible failures? What could you do to handle them?

Server could go offline causing the client(s) to wait endlessly for at response. This could be addressed by telling the client that the server disconnected. This could be detected by a limit on how long the client should at most wait for a response.

The service is reliant on a local network with static IP-adresses. If the adresses somehow change during send-receive, the reply might go to the wrong client.

Messages larger than the buffer size will be truncated and therefore loose data.

Messages is trimmed, so leading or trailing whitespace will be disregarded.

8

UDPClient

```
import java.net.*;
import java.io.*;
import java.util.Scanner;

public class UDPClient {
   private static String serverIP = "10.26.9.128"; // Kristoffers IP
   private static int incomingPort = 1337;
   private static int outgoingPort = 7007;

   public static void main(String args[]) {

       DatagramSocket socket = null;
       try {
           socket = new DatagramSocket(incomingPort);
       } catch (Exception e) {
           e.printStackTrace();
           return;
       }

       Scanner msgScan = new Scanner(System.in);

       try {
           while (true) { // Keep asking user for messages.
               System.out.println("Type a message..");

               // Read a message from standard input
               String msg = msgScan.nextLine();
               byte[] msgBytes = msg.getBytes();

               // Send the message

               InetAddress aHost = InetAddress.getByName(serverIP);
               DatagramPacket request = new DatagramPacket(msgBytes, msgBytes.length, aHost,
                    outgoingPort);
               socket.send(request);

               System.out.println("Message sent. Waiting for reply from");

               System.out.println(aHost);

               // Receive reply
               byte[] buffer = new byte[1000]; // Allocate a buffer into which the reply
                   message is written
               DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
               socket.receive(reply);

               // Print reply message
               System.out.println(
                  "Received reply: \"" +
                  new String(reply.getData()).trim() +
                  "\""
               );

           }
       } catch (SocketException e) { // Handle socket errors
           System.out.println("Socket exception: " + e.getMessage());
       } catch (IOException e) { // Handle IO errors
           System.out.println("IO exception: " + e.getMessage());
       } finally { // Close socket
           if (socket != null)
               socket.close();
       }
   }
}
```

UDPServer

```
import java.net.*;
import java.io.*;

public class UDPServer {
    private static int outgoingPort = 1337;
    private static int incomingPort = 7007;

    public static void main(String args[]) {
        System.out.println("Server listening on port " + incomingPort);

        DatagramSocket socket = null;
        try {
            socket = new DatagramSocket(incomingPort);
        } catch (Exception e) {
            e.printStackTrace();
            return;
        }

        try {

            while (true) { // Keep asking user for messages.
                // Receive reply
                byte[] buffer = new byte[1000]; // Allocate a buffer into which the reply message
                DatagramPacket incomingMessage = new DatagramPacket(buffer, buffer.length);
                socket.receive(incomingMessage);

                // Print reply message
                System.out.println(
                    "Received: \"" +
                    new String(incomingMessage.getData()).trim() +
                    "\" from " +
                    incomingMessage.getAddress()
                );

                // Read a message from standard input
                String msg = new String(incomingMessage.getData()).trim();
                byte[] msgBytes = msg.getBytes();

                // Send the message
                InetAddress replyAddress = incomingMessage.getAddress();
                DatagramPacket request = new DatagramPacket(
                    msgBytes,
                    msgBytes.length,
                    replyAddress,
                    outgoingPort
                );
                socket.send(request);

                System.out.println(
                    "Send back: \"" +
                    new String(incomingMessage.getData()).trim() +
                    "\" to " +
                    replyAddress
                );

            }
        } catch (SocketException e) { // Handle socket errors
            System.out.println("Socket exception: " + e.getMessage());
        } catch (IOException e) { // Handle IO errors
            System.out.println("IO exception: " + e.getMessage());
        } finally { // Close socket
            if (socket != null)
                socket.close();
        }
    }
}
```