Assignment 2
Topic: Socket Programming

Exercise 2.1 – Java UDP 20%

Explain the fundamental differences between TCP and UDP and when you would use which of the two protocols. Demonstrate how UDP communication can be accomplished in Java by implementing a simple client-server application *Quotation*. The application should work the following way:

1. The client sends an empty message to the server.
2. The server responds with a random quote (use a list of one-liners as basis)
3. The client prints the quote

Exercise 2.2 – Simple File Server 40%

Write a rudimental file server that makes a collection of files available for transmission, as well as an according client that uses this service. The server needs to know the name of the directory that contains the collection of files and the port on which it should listen. Provide this information as command-line arguments. You can assume that the directory contains only regular files of any type (e.g., text files, images, videos, but no sub-directories).

The client can send two different commands to the server:

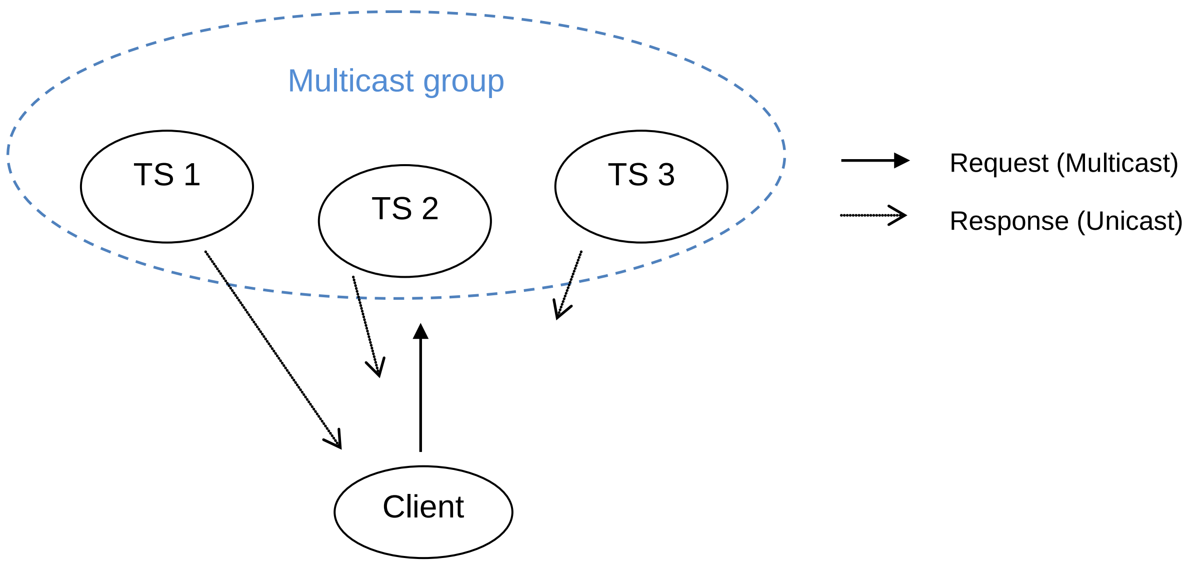
- “list”: the server responds with a list of all file names in the observed directory.
- “get <file>”: After checking if <file> exists in the observed directory, the server reads the (binary) file content and transmits it to the client. The client reads the binary data from the stream and writes it into a file. When the file transfer is completed, the connection is closed. In case <file> is no valid filename, the server should send some kind of an error message.

The server should start a separate thread to handle each connection request. The client should receive user input form the keyboard.

Hint: until now we only sent text messages using PrintWriter and BufferedReader. For transmission of binary data (like a file), you should rather write and read the according bytes directly to/from the socket input/output stream. It might be helpful for the client to first receive information about how many bytes will be sent.

Exercise 2.3 – Distributed Time server **40%**

Use Java Sockets and DatagramPackets to implement a distributed time server system that consists of several time servers and requesting clients. More precisely, several time servers form an IP-Multicast group that can be used by clients to request the current time. Such a request triggers all time servers in the group to respond to the client with a Unicast message. The requesting client should use the average time value received within one second. A time server has a manually assigned unique identifier and joins the multicast group at startup (it should be possible to start multiple time servers).



Please use the following conventions for communication:

- A request message contains the string "`REQ`" followed by the client i.e. destination port number the server should use for the response (use # as a delimiter).
  Example: `REQ#7777#`
- A response message contains the server ID followed by its local time in milliseconds (as returned by java.util.Date.getTime()).
  Example: `TS-1#1178530662375#`