- Client/Server-Example with Multi-Threading (Bonus) -

Goal:
In this exercise you will expand your knowledge in the are of multi-threaded Client/Server applications. You will extend the single-threaded C/S example by a multi-threaded architecture similar to the exercise on multi-threading.

Exercise 05a.1: Create some real workload to get started
Why do we need a multi-threaded server? To better understand this, you first copy the echo server into a new Eclipse project and then you create some workload in the server while responding to the client request (i.e. doing some computation loops). What happens when another client request (or several other requests) hits the server while it is still busy working on the current request?

Exercise 05a.2: Dynamic creation of threads
Now modify the server so its main loop only waits(!) for client requests. In advent of a new request, the server creates and starts a new thread on demand, which implements the application protocol (i.e. the echo functionality). Pls. note that in this case the default port might already be busy. To solve this problem, the server will allocate the “next available” port to the client. Only this port is used for the exchange of the echo message. Extend the application protocol between client and server accordingly.

These threads are called “worker threads”, because they execute the “real” work. Worker threads shall improve the response time of the server in case of multiple parallel client requests, because they avoid blocking. Do not forget to clean up the worker threads after they have finished. When executing the load test from exercise 051.1: which behavior do you observe?

Exercise 05a.3: Now we get really „professional“
In real life situations, even creating and cleaning up of dynamic threads might take too much time blocking the server. So it is reasonable, in case of an average of n clients, to create a set of n threads (or even 2*n just for safety reasons) a priori, which are just sitting “idle” until a request arrives. After completing its work, each thread will become idle again. This architecture is called a “thread pool”.

Based on the results of 05a.2, create such a thread pool on start-up of the server. In case, all threads are busy and new client requests still arrive, you can dynamically double the size of your thread pool or reject the request due to “server busy…” .