Task 2: Network of Counters – the UDP variant
SKJ (2017)

Introduction

Precise measurement of time is of utmost importance both in science and technology. The task is not easy, in particular due to the fact that it is difficult to synchronise clocks that are set apart from each other by distance and/or signal travel time. The second project deals with a simple model of synchronisation of distributed network of counters based on UDP connections.

Specification

Agent

The **agents** deal with counters and connections. Each agent is a separate node, identified by its **IP address** (due to technological limitations, there may be only one agent per physical node). The IP of an agent depends on the node on which the agent runs. The port number is the same for all agents. This port is used by an agent for communication.

Each agent has a **single counter** that measures time (in milliseconds) and initially is set to a value, which is a runtime parameter.

Each agent periodically, every specified time period, synchronizes its cunter value with values of counters of other agents.

Agent interactions

Adding agents

All agents work according to the following scheme:

1. An agent is executed with 2 parameters:
   (a) initial counter value for this agent,
   (b) time period (in seconds), which denotes the interval between synchronization procedure done by an agent.

2. After execution, each agent periodically, every required time period, synchronises its counter with values of counters of other agents.

3. An agent may end its wotk at any moment. If needed, it must notify other agents about this fact (if required by the assumed communication scheme).

Synchronisation of counters

Each of the agents periodically synchronises its counter with all other counters working in its network. It can use the broadcast:

1. Agent i sends a broadcast message with request of delivery of a current counter state to all agents, who receive this message.

2. All agents, who receive such message, must respond sending back their current counter values.

3. The agent i gathers all the responses (one should expect unknown number of responses, adequate to the number of currently running agents).

4. The agent i sets the new value of his counter as an average from the obtained values (including his own, i.e., T_i):
$$T_i := \frac{1}{N} \sum_{j=1}^{N} T_j.$$

5. The agent i prints its current counter value to its output.

The controller

An additional application is used as a controller of the network of agents. Using it, one can read or change the value of the counter or the synchronization period of each agent separately. The application as its first parameter acepts the the IP address of an agent (the port number is the same for all agents). The rest of parameters depends on the requested operation:

- For reading: 2 parameters:
  1. the second parameter is a keyword `get`,
  2. the third parameter is either `counter` or `period`, depending on a value to be read.
- For writing: 3 parameters:
  1. the second parameter is a keyword `set`,

2. the third parameter is either `counter` or `period`, depending on a value to be changed.
3. the fourth parameter is the new value.

After execution, the controller application sends a message to the requested agent and then waits for confirmation. After an agent gets a request, stores the obtained value and sends back acknowledgement (if `set`) or returns the requested value (if `get`).

Grading and requirements

1. Fully and correctly implemented project is worth **4 points**. Implementing each of the following functionalities is rewarded up with the specified number of points:
   - Agent network creation and correct counter synchronization: **2 points**.
   - The `get` function of the controller: **1 point**.
   - The `set` function of the controller: **1 point**.
2. The program should be written in Java, in compliance with Java 8 standard (JDK 1.8). Only basic TCP socket classes can be used for implementing the network functionality. A different language can be used with permission of teaching assistant.
3. The projects should be saved in appropriate directories of EDUX system not later than on 22.XII.2017 (the deadline can be moved by the teaching assistant).
4. The archived project file should contain:
   - File *Documentation(indexNo)Task2.pdf*, describing what has been implemented, what hasn't, what were the difficulties and what errors are still present.
   - Source files (JDK 1.8) (together with all the libraries used by the author that are not the part of Java standard library). The application should compile and run on PJA lab's computers.

   NOTICE: THE DOCUMENTATION FILE IS NECESSARY. PROJECTS WITHOUT DOCUMENTATION WILL NOT BE GRADED.
5. Solutions are evaluated with respect to the correctness and compliance with the specification. The quality of code and following software engineering rules can in uence the final grade.
6. UNLESS SPECIFIED OTHERWISE, ALL THE AMBIGUITIES THAT MIGHT ARISE SHOULD BE DISCUSSED WITH TEACHING ASSISTANT. INCORRECT INTERPRETATION MAY LEAD TO REDUCING THE GRADE OR REJECTING THE SOLUTION.