Task 1: Network of Counters
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 first project deals with a simple model of synchronisation of distributed network of counters based on TCP connections.

Specification

Agent

The **agents** deal with counters and connections. Each agent is a separate node, identified by **IP address** and **port number**. The port is used by the agent to communicate with the other agents and the monitor (introduced later). Each agent has a **single counter** that:

- Measures passage of time (in milliseconds).
- Can be set to any value.

Additionally, every agent **knows the full structure of network**, i.e., knows IP addresses and communication ports of all the remaining agents. To this aim, the agent can keep connection to the other agents open or collect the data in offline record. Each of the agents can be requested to provide the current value of the counter and the description of the network by sending the following communicates to his listening port:

- **CLK** - the agent sends back the current value of the clock.
- **NET** - the agent sends back the record of IP numbers and communication ports of all the agents.

Agent interactions

Adding agents

Each agent is provided with the initial value of his counter as an argument (an possibly port value). The first agent is instantiated without any additional parameters, becoming the first node of the network; after initialisation the agent's counter is started. Each following agent is provided with IP addres and port number of **introducing agent**, already present in the network, as additional parameters.

After connecting to his introducing agent, the agent downloads from him the details of all the remaining ones. Then, he provides to each of the agents his IP address and port number. Afterwars, the agent performs **synchronisation step**, as described in the next section.

Synchronisation of counters

Each of the agents can be requested to synchronise his counter by sending **SYN communicate** to **the agent's port**. The process of synchronisation is as follows:

1. Agent i downloads from all the agents the values of their counters $T_1, \ldots, T_N$.
2. Agent i sets the new value of his counter as an average from the obtained values (including his own, i.e., T_i):
$$T_i := \frac{1}{N} \sum_{j=1}^{N} T_j.$$

Additionally, an agent performs a synchronisation step when connecting to network, as described above. In this case, after setting his counter to a new value, he sends to the other agents SYN communicate.

Example

Let us consider a network with agents A and B. Agent C connects to the network and agent A is his introducing agent. An exemplary interaction sequence is as follows:

1. Agent C downloads from A his list of contacts.
2. Agent C sends his data to A and B, who update their contact lists.
3. Agent C downloads from A and B the values of the counters and updates his counter with a new value.
4. Agent C sends communicate SYN to agents A and B.
5. Agent A receives SYN and reacts by obtaining counter values from B and C and updating his counter.
6. Agent B receives SYN and reacts by obtaining counter values from A and C and updating his counter.

Monitor

The monitor is a simple program that displays the state of the network of counter. It should present a table with the list of agents (i.e., pairs IP:port) together with their counter values. The table should be dynamically updated. The monitor can be either a console program or a http server (additional points). In the latter case, it should allow for sending SYN communicate to any agent (**rewarded by an additional point**) and for disconnecting an agent from the network (**rewarded by an additional point**). Of course, after disconnecting, all the remaining agents should update their contact lists and re-synchronise their clocks.

Grading and requirements

1. Fully and correctly implemented and documented project is worth **5 points** (2 additional points can be obtained by implementing optional functionality). Implementing each of the following functionalities is rewarded up with the specified number of points:

   - Correctly adding the agents to the network, exchanging connection lists. **2 points**.
   - Correct synchronisation of counters. **2 points**.
   - Basic monitor. **1 point**.
   - Extended monitor (see above). **two points**

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 26.XI.2017 (the deadline can be moved by the teaching assistant).

4. The archived project file should contain:

   - File *Documentation(indexNo)Task1.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 influence 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.