Programming task, SKJ – winter semester 2022/23

Subect: **A distributed database**

Description:
A distributed database consists of a set of processes located in the network. Each process hosts a certain set of information (for simplicity we assume that it’s just one pair <key,value> per process). Processes constitute a logical network, in which each process is connected with at leas one other process in a network. The network is incrementally created by executing consecutive processes (creation of consecutive nodes) and connecting them to the already existing network. After starting, each process (except for the first one) connects to the network according to the assumed network communication model and starts waiting for possible connections from new system components (next added nodes) or from clients willing to reserve the resources. Communication with clients and newly connected nodes uses the TCP protocol. Communication between the nodes during the database operation can use both TCP and/or UDP – it’s left to the programmer. All database nodes are symmetric, i.e. they both keep their data and accept requests from clients. A node may communicate only with the nodes to which it connected at the start or nodes which connected to it later.

The network node execution has the following form:

```
java DatabaseNode -tcpport <numer portu TCP> -record <klucz>:<wartość>
     [ -connect <adres>:<port> ]
```

where:
- `-tcpport <TCP port number>` denotes the number of the TCP port, which is used to wait for connections from clients.
- `-record <key>:<value>` denotes a pair of integers being the initial values stored in this node, where the first number is the key and the second is a valu associated with this key. There is no uniqueness requirement, both to a key and a value.
- `[ -connect <adres>:<port> ]` denotes a list of other nodes already in the network, to which this node should connect and with which it may communicate to perform operations. This list is empty for the first node in the network.

Execution example:

```
java DatabaseNode -tcpport 9991 -record 17:256 -connect localhost:9990
     -connect localhost:9997 -connect localhost:9989
```

The command means execution of a node which holds a value 256 associated with a key 17, listens to the TCP port 9991 for connections from clients and other nodes willing to connect and to connect itself to the network it should use the nodes working on a computer with an address `localhost` and TCP ports 9990, 9997 and 9989.

After the network is created, clients may be connected to it. A client is a simple process, which goal is to send its request to the network in a proper format and wait for a response. During execution, a client obtains as a parameter an IP address of one of the network nodes (any of them, called later the “contact node”), the TCP port number on which this node waits for client connections and an operation to be performed (in a text form). A client execution has the following form:

```
java DatabaseClient -gateway <adress>:<TCP port number>
     -operation <operation with parameters>
```

Exemplary client execution:

```
java DatabaseClient -gateway localhost:9991 -operation get-value 17
```

denotes execution of a client, which will use the node working on the machine with address `localhost` and TCP port number 9991 for connection with the network. After it connects, the database should perform a search operation for a value assigned to the key 17.

After a client connects with the network node, it sends a single line message in the format:

```
<operation> [ <parameter> ]
```

(being the text after the `-operation` parameter), where consecutive elements depend on the operation to be performed. The available operations together with parameters look as follows:

1. `set-value <key>:<value>` : set a new value (the second parameter) for the key being the first parameter. The result of this operation is either an `OK` message if operation succeeded or `ERROR` if the database contains no pair with a requested key value. If the database contains more than 1 such pair, at least one must be altered.
2. `get-value <key>` : get a value associated with the key being the parameter. The result of this operation is a message consisting of a pair `<key>:<value>` if operation succeeded or `ERROR` if the database contains no pair with a requested key value. If the database contains more than 1 such pair, only one pair must returned (any valid).
3. `find-key <key>` : find the address and the port number of a node, which hosts a pair with the key value given as the parameter. If such node exists, the answer is a pair `<address>:<port>` identifying this node, or the message `ERROR` if no node has a key with such a value. If the database contains more than 1 such node, only one pair must returned (any valid).
4. `get-max` : find the biggest value of all values stored in the database. The result is a pair consisting of `<key>:<value>`. If the database contains more than 1 such pair, only one pair must returned (any valid).
5. `get-min` : find the smallest value of all values stored in the database. The result is a pair consisting of `<key>:<value>`. If the database contains more than 1 such pair, only one pair must returned (any valid).
6. `new-record <key>:<value>` : remember a new pair key:value instead of the pair currently stored in the node to which the client is connected. The result of this operation is the `OK` message.
7. `terminate` : detaches the node from the database. The node must inform its neighbours about this fact and terminate. The informed neighbours store this fact in their resources ans no longer communicate with it. Just before the node terminates, it sends back the `OK` message to a client.

Each request consists of exactly one line in a format as described above ended with a new line character. A network node after it receives the request, attempts to perform it. In particular, operations 1-5 require network communication to find a node responsible for a record, operation 6 requires only communication with the node to which a client is connected and operation 7 – communication with the neighbours. The way in which operations are performed and communication scheme is left to the programmer – although it is forbidden to use a centralised model with a central server having knowledge about all records in the network. The result of each operation is a single line ended with a new line character. After the response is obtained, a client prints it on a screen and terminates.

Remarks:

- On a single physical machine many logical network nodes may run. Therefore the communication ports must be properly managed to avoid conflicts.
- The nodes performing operations may print to their consoles messages about their progress (for instance, who is requesting what and so on).
- If a value change operation fails, the system must stay in the same state in which it was before this allocation attempt.

- The project must unconditionally obey the main class naming rule (like in examples above) and the way in which the application is started. We will use an automatized system for testing and no differences will be accepted.
- There's no need to write a client – it will be delivered by us at the beginning of January.

Specification:

1 The task consists in design and implementation of the protocol and an application using it, which will allow for organization of a distributed database and further operation on it – as described above. The way in which the node network is organized, connection of new nodes to an already existing network and methods for their communication towards operation execution are elements of the task.

2 Project solutions must be saved to proper directories (in the Tasks section) on the GAKKO system before 15.01.2023 (the date may be changed by the teacher of a group).

3 For a correct solution of the task a student may obtain up to 500 points:

3.1 At most 200 points for a complete documentation describing the way in which the network is created, organized, description of the application communication together with description of messages and communication protocol. **Presence of a complete documentation is required for the next components of this project to be checked** (3.2, 3.3 i 3.4)**.**

3.2 Up to 100 points for implementation of a network node process, which allows for organization of such network (connection of additional nodes and termination).

3.3 Up to 100 additional points for implementing in a process functionality required to accept requests from clients and performing operations under an assumption, that only one client operates on a database at a time (also requires implementation of 3.2).

3.4 Up to 100 additional points for implementing the system allowing for concurrent work of more than one client at a time (also requires implementation of 3.2 and 3.3).

4 The project archive must contain:

4.1 Source files (for JDK 1.8)

4.2 Binary (class) files,

4.3 Scripts for Linux (and/or Windows system) which allow to compile and run the proposed solution (from a command line, without using GUI).

4.4 The readme.txt file with author's description and remarks, especially:

4.4.1 **Detailed description of the implementation** (no description or incompleteness of protocol description may significantly reduce the grade), especially with the detailed description of the communication protocol (message formats, message handling etc. required for 3.1).

4.4.2 how to compile/install from a command line not using a GUI,

4.4.3 what was implemented,

4.4.4 what does not work (if anything).

5 IF NOT DETAILED, ALL UNCERTAINTIES SHOULD BE DISCUSSED WITH THE TEACHER. OTHERWISE, THE SOLUTION MAY BE NOT ACCEPTED, IN CASE OF WRONG SELF-INTERPRETATION.