COMP 6231: Distributed Systems Design

Assignment 1 on Sockets and Peer Discovery

Summer 2022, sections BB

May 9, 2022

Contents

1 General Information

Date posted: Tuesday, May 10th, 2022.

Date due: Monday, May 30th, 2022, by 23:59.[1].

Weight: 5% of the overall grade.

2 Introduction

This assignment targets network communication and peer discovery using java sockets[2].
In this assignment you implement a small distributed repository system using dictionaries and sockets. The function and architecture of the system is described in section 4.

3 Ground rules

You are allowed to work on a team of 2 students at most (including yourself). Each team should designate a leader who will submit the assignment electronically. See Submission Notes for the details.
ONLY one copy of the assignment is to be submitted by the team leader. Upon submission, you must book an appointment with the marker team and demo the assignment. All members of the team must be present during the demo to receive the credit. Failure to do so may result in zero credit.
This is an assessment exercise. You may not seek any assistance from others while expecting to receive credit. **You must work strictly within your team).** Failure to do so will result in penalties or no credit.

[1] see submission notes

[2] The Official supported programming language for this course is Java, however, you may implemented the system using C, C#, Python, or any programming language of your choice.

4 Overview

In this assignment you implement a small distributed repository. The distributed repository system in this assignment consists of one or more independent peers that may run on a single or different IPs over on local network. Detailed high-level specification[3] of the system is given in the following.

The system speciation is given in two parts: (i) Single Repository, and (ii) Distributed Repository.

4.1 Part I - Single Repository

The following specifies the functions of a repository, to be implemented on a single peer.

4.1.1 Repository Functions

Each peer is designed as an independent repository, implementing a dictionary that is storing values associated with a unique key. Duplicates are allowed, by which, a collection of values is associated with the key.

The repository functions are listed in the following:

1. Add a value: upon existence of the key, the value is added to a collection.
2. Set (Update) a value: key must exist; previous value(s) is/are overwritten
3. Delete a value: upon existence of the key, the key-value entry is removed from the dictionary.
4. List keys.
5. Get value: upon existence of the key, returns any (i.e. first) value that is associated with the key. If key does not exist, returns error.
6. Get values: return all values associated with the key, in a comma separated format[4].
7. Aggregates: implement some aggregates of your choice: i.e. max, min, sum (numbers only), avg (numbers only), etc. that aggregates all elements associated with a key and returns the result.
8. Reset: delete all keys.

[3] These are indeed requirements, to be translated into design specifications, as part of the assignment.
[4] If entries contain quote, properly escape them

4.1.2 The RAP Protocol

The Repository Access Protocol (RAP) (to be designed and implemented) specifies how a directory entry may be accessed and modified remotely. You may designed and implement your own protocol, text based (as used in the labs) or in binary.
An example is given in the following.

```
SERVER: OK Repository <<ID>> ready
CLIENT: SET a 12
SERVER: OK
CLIENT: DELETE b
SERVER: OK
CLIENT: ADD a 13
SERVER: OK
CLIENT: GET a
SERVER: OK 12, 13
CLIENT: SUM a
SERVER: OK 25
CLIENT: SET a 10
SERVER: OK
CLIENT: GET a
SERVER: OK 10
```

4.1.3 IDs and Listening Ports

Each peer is identifies by an ID, runs on a TCP port of a choice and uses an in-memory key-value dictionary to store values associated with keys. The ID associated with a peer is to be considered unique[5].

4.1.4 The Client

The client uses the protocol in 4.1.2 to connect to a repository that is listening on a port that is entered by the user. There is no need to implement a client application. You may use *telnet* or *netcat* to connect to your repository server(s) and test their functionalities.

[5]See peer discovery notes in 4.2.2

4.2 Part II - Distributed Repository

This sections specifies how the collection of peers may be used to implemented a distributed repository.

4.2.1 Peers Posts and Locations

Peers can run on a single or multiple IP/ports on the local network. There is no assumption on the number of concurrent peers as well as the listening port.

4.2.2 Peer Discovery

Since location and listening ports of the peers are unknown, a peer discovery protocol (PDP) is to be implemented. You may implement the PDP protocol using UDP and broadcast addressing, either by advertising mechanism, or by inquiring existing peers on a local network. The Peer Discovery Protocol may be used the client or other repositories in the system. See 4.2.5 for more details.
You may use UDP on a known port to implement your own PDP protocol. Note that multiple peers may be run on a single IP address [6].

4.2.3 The Client

The client may connect to any repository and access all repositories by using the ERAP protocol, as specified in 4.2.4. You may also use *telnet* or *netcat* to simulate the client. Implementing the client application is optional. See also section 5 for sample input/output.

4.2.4 The ERAP Protocol

This section extends the RAP protocol to allow distributed repository access. We use a dotted naming convention to access keys on various repositories. We assume that keys cannot contain dots.

[6] See 5 to run multiple listeners on a single port.

Example: Suppose `akey` represent a key in the local repository. `repid.akey` then represents the key "akey" in the remote repository identified by "repid". See 4.2.6 for more details.

4.2.5 Directory Access and Repository Proxy

The ERAP protocol enables accessing remote repositories by using the naming convention, specified above (section 4.2.4). If no "repid" is provided in the key, the protocol behaves as a single repository. Upon receiving a "repid" in the key, the peer acts as a proxy and forwards the client request to the remote peer and reports back the result to the client. This requires the following two steps:

1. The peer removes the repid, when passing the requests to remote repository. The peer must also check whether the repid is referring to self.
2. The peer must implement a peer discovery, to locate the peer on the network and record keeps track of the listening port. In case two or more peers use a duplicate repid (which is an error), the discovery fails with the following message to be reported to the user "Non-existence or ambiguous repository «REPID»".

4.2.6 Extended Functionality

The ERAP protocol implements applying aggregates on multiple repositories. An example is given in the following.

```
CLIENT: SET a 25
SERVER: OK
CLIENT: SET r2.a 1000
SERVER: OK
CLIENT: SET r3.a 100
SERVER: OK
CLIENT: SUM a
SERVER: OK 25
CLIENT: DSUM a INCLUDING r2 r3
SERVER: OK 1025
CLIENT: DSUM a INCLUDING r2 r3 r4
```

```
SERVER: ERR Non-existence or ambiguous repository r4
```

The `INCLUDING` directive in the first example tells the server that the `DSUM` aggregate is applied on local and the two remote peers identified by "r2" and "r3" .

5 Your Assignment

Your assignment is composed of the following components:

1. The Repository Core: the business, as specified in 4.1.1
2. The Repository Server: the TCP listener
3. The Peer Discovery Module: as specified in section 4.2.2. It may optionally be used in the client program to give the user a list of repositories to choose from.
4. The repository client: the client program that connects to a given repository, implementing the client side protocol as shown in sections 4.1.2 and 4.2.4. The client may be interactive or a program that reads the content of a file and sends the commands to a remote repository to execute.
5. A sample input / output: showing how the RAP and ERAP protocols are implemented.

Implementation Platform: The recommended implementation platform is Java. However, you may use any platform or programming language of your choice.

Source Control: You may use GitHub or an alternative source control during development phase. Using a source control software is mandatory.

UML Diagram: Upon completion of your project, generate and include the "UML class / package diagram" of all implemented classes in your project.

Sample I/O: Create a file and put some client commands in the file. Pass all commands the a repository server and report the outputs on the screen.
Note that there is no need to write a client application. If you wish to do so, make sure your program an read the commands from a file. You may run *telnet* or *netcat* (see 6) and redirect the input and output to / from files.

```
cat intput.txt | nc localhost 6231 > output.txt
```

6 What to Submit

The whole assignment is submitted by the due date under the corresponding assignment box. It has to be completed by ALL members of the team in one submission file.

Submission Notes

Clearly include the names and student IDs of all members of the team in the submission. Indicate the team leader.

IMPORTANT: You are allowed to work on a team of 2 students at most (including yourself). Any teams of 3 or more students will result in 0 marks for all team members. If your work on a team, ONLY one copy of the assignment is to be submitted. You must make sure that you upload the assignment to the correct assignment box on Moodle. No email submissions are accepted. Assignments uploaded to the wrong system, wrong folder, or submitted via email will be discarded and no resubmission will be allowed. Make sure you can access Moodle prior to the submission deadline. The deadline will not be extended.

Naming convention for the uploaded file: Create one zip file, containing all needed files for your assignment using the following naming convention. The zip file should be called a#_studids, where # is the number of the assignment, and studids is the list of student ids of all team members, separated by (_). For example, for the first assignment, student `12345678` would submit a zip file named `a1_12345678.zip`. If you work on a team of two and your IDs are `12345678` and `34567890`, you would submit a zip file named `a1_12345678_34567890.zip`.

Submit your assignment electronically on Moodle based on the instruction given by your instructor as indicated above: `https://moodle.concordia.ca`

Please see course outline for submission rules and format, as well as for the required demo of the assignment. A working copy of the code and a sample output should be submitted

for the tasks that require them. A text file with answers to the different tasks should be provided. Put it all in a file layout as explained below, archive it with any archiving and compressing utility, such as WinZip, WinRAR, tar, gzip, bzip2, or others. You must keep a record of your submission confirmation. This is your proof of submission, which you may need should a submission problem arises.

7 Grading Scheme

The Repository Core	10 marks
The Repository Server	20 marks
The RAP Protocol	20 marks
Peer Discovery and the PDP Protocol	15 marks
Aggregation	10 marks
Directory Access and Proxy	10 marks
The UML Diagram	10 marks
The Sample Input/ Output	5 marks

Total: 100 marks.

References

1. Key-Value Database: `https://en.wikipedia.org/wiki/Key%E2%80%93value_database`
2. Java Sockets: `https://www.baeldung.com/a-guide-to-java-sockets`
3. A Guide To UDP In Java: `https://www.baeldung.com/udp-in-java`
4. Java NIO Sockets:

   `https://www.tutorialspoint.com/java_nio/java_nio_socket_channel.htm`
5. **DatagramSocket.setReuseAddress()**:

   `https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html`
6. NetCat: `http://netcat.sourceforge.net/`
7. Java Concurrency:

   `https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html`