TCSS 558: Applied Distributed Computing
School of Engineering and Technology
Winter 2023
University of Washington – Tacoma
http://faculty.washington.edu/wlloyd/courses/tcss558
Instructor: Wes Lloyd

Assignment 1
Version 0.10
Key Value Store – Client/Server

Due Date: Tuesday February 21st, 2023 @ 11:59 pm, tentative

Objective

The purpose of assignment 1 is to build socket-based and/or remote-object based clients and servers. For assignment 1 produce a program called "GenericNode". GenericNode will receive command arguments and then take on the role of a client or server for a basic Key-Value store. The following operations should be supported:

Operation	Description
put	put <key> <value> Put a value to the key-value store. The key uniquely identifies the object to store. A unique key maps to only one object which has a unique value. If multiple clients write to the same key at the same time, writes should be synchronized so that only one client can change a key at any given time. If the server uses only 1-thread, no synchronization is necessary. If the server is multi-threaded, two concurrent writes to the same key should not be allowed. This could result in a non-deterministic ordering of changes. One thread should write first (based on FCFS), while others block and wait and before proceeding. A blocked write will proceed based on the order of requests when the key is available.
get	get <key> Returns the value stored at <key>.
del	del <key> Deletes the value stored at <key> from the key value store.
store	store Prints the contents of the entire key-value store. You may optionally truncate the output after returning 65,000 characters. If the contents of the key value store exceed 65,000 bytes, the return should start with "TRIMMED:" followed by the content.
exit	exit When the exit command is sent by the client, the server is shutdown.

For the GenericNode program, the goal is to first implement a TCP client and server. Then the GenericNode program should be expanded to implement a client/server using additional protocols: UDP (connection-less), and remote objects (Java RMI).

The end goal is to perform a performance comparison of the transaction execution time for at least two of the three protocols: TCP, UDP, and RMI.

Students who submit a working GenericNode and performance comparison using all three protocols (TCP, UDP, and RMI) without significant errors will be eligible for **up to 20% extra credit**.

The preferred implementation for assignment #1 is in Java 11. Students can implement assignment #1 in Python if needed. Implementation in other languages requires instructor approval. Assignments in other languages without prior instructor approval will not be graded.

Submissions in Python and Alternate Languages

Solutions in languages other than Java must include detailed documentation describing how to deploy and operate the client and server. All operations including setup must be explained. If the operation of any client or server functions (put, get, del, store, exit) is unclear, no credit will be granted for these operations. Solutions with insufficient documentation will be returned with no grade. For extra credit, distributed objects in Python can use Pyro remote objects (4.x) available here: **https://pyro4.readthedocs.io/en/stable/**. Python implementations using Pyro must include documentation to explain precisely how to deploy and test the system. The key is that CLEAR instructions are needed. If substantial additional effort is required to decipher and test the implementation due to insufficient documentation, expect to receive a documentation penalty (point deduction), or worse, no grade at all.

Docker for Assignment #1

Dockerfiles should be provided that encapsulate the GenericNode client and server to aid in testing the project. The project submission should include a **docker_client** and **docker_server** directory that includes all of the required files to build a client and server docker container (i.e. Dockerfile, jar file, other required files). These files may be referred to and used to help test the project.

Clients and servers deployed using Docker containers should be able to communicate using the local subnetwork shared among containers of a single Docker host. Docker will dynamically assign local IP addresses to each container. The client container will use the docker container private network IP address to facilitate communication with the server. If not using Docker containers, then clients and servers can communicate on the local computer using the localhost (127.0.0.1) and unique port numbers.

To support working with Docker containers, Dockerfiles for the client and server have been provided. ==**Sample docker_client and docker_server files have been added to the GenericNode.tar.gz file that is downloaded on page 8.**== *Feel free to use these, or develop new Dockerfiles...*

When downloading GenericNode.tar.gz, "cd" into the individual docker_server or docker_client directories to build the docker images.

The sample dockerfiles includes a placeholder indicating where the GenericNode.jar Java archive file created using Maven should be copied. For assignment #1, compile the GenericNode.jar using Maven, and copy the file to the docker_client and docker_server directories.

Inside the docker_server directory, a runserver.sh script has been provided.
This script includes a command to start a server for each of the given types.

When building your docker_server container, uncomment the specific server you'd like to run: TCP, UDP, or RMI. Remove the "#":

```
# Dummy jar file
java -jar GenericNode.jar

#TCP Server
#java -jar GenericNode.jar ts 1234

#UDP Server
#java -jar GenericNode.jar us 1234

#RMI Server
#rmiregistry -J-Djava.class.path=GenericNode.jar &
#java -Djava.rmi.server.codebase=file:GenericNode.jar -cp GenericNode.jar genericnode.GenericNode
rmis
```

Once running, to discover the internal IP address of your server running on a Docker host, use the following sequence:

First, build the docker_server container:

```
$ cd docker_server
$ sudo docker build -t tcss558server .
Sending build context to Docker daemon   5.12kB
Step 1/7 : FROM ubuntu
 ---> ccc7a11d65b1
Step 2/7 : RUN apt-get update
 ---> Using cache
 ---> 1413c1a1f91b
Step 3/7 : RUN apt-get install -y default-jre
 ---> Using cache
 ---> b23e154d7af3
Step 4/7 : RUN apt-get install -y net-tools
 ---> Using cache
 ---> 1d81d5652fc2
Step 5/7 : COPY GenericNode.jar /
 ---> Using cache
 ---> f74d73c86c5c
Step 6/7 : COPY runserver.sh /
 ---> Using cache
 ---> f23167bd7d09
```

Step 7/7 : ENTRYPOINT /runserver.sh
 ---> Using cache
 ---> e921fbb5db7a
Successfully built e921fbb5db7a
Successfully tagged tcss558server:latest

Then, run the docker container:

$ sudo docker run -d --rm tcss558server
1ad8abcb16cae530322464099487d028154a2452072e5e20f6007ff3e5f1a66d

Now, grab (copy and paste) your unique CONTAINER ID. The Name can also be used (here distracted hodgkin):

```
$ sudo docker ps -a
CONTAINER ID      IMAGE          COMMAND          CREATED        STATUS          PORTS        NAMES
1ad8abcb16ca      tcss558server  "/runserver.sh"  4 seconds ago  Up 4 seconds                 distracted_hodgkin
```

Next, execute bash interactively on this container

$ sudo docker exec -it 1ad8abcb16ca bash

Then, use the "ifconfig" command inside the container to query the local IP address:

```
root@1ad8abcb16ca:/# ifconfig
eth0    Link encap:Ethernet  HWaddr 02:42:ac:11:00:02
        inet addr:172.17.0.2 Bcast:0.0.0.0  Mask:255.255.0.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:48 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:6527 (6.5 KB)  TX bytes:0 (0.0 B)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        UP LOOPBACK RUNNING  MTU:65536  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1
        RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

It is recommended to develop clients and servers on a local machine, and then TEST your program using deployments to Docker containers before submitting.

Running the servers

Servers should support the following syntax:

```
#TCP Server (ts for TCP server)
java -jar GenericNode.jar ts <server port number>

#Example:
java -jar GenericNode.jar ts 1234

#UDP Server (us for UDP server)
java -jar GenericNode.jar us <server port number>
#Example:
java -jar GenericNode.jar us 1234

#RMI Server (rmis for RMI server)
rmiregistry -J-Djava.class.path=GenericNode.jar &
java   -Djava.rmi.server.codebase=file:GenericNode.jar   -cp   GenericNode.jar
genericnode.GenericNode rmis
```

For Assignment #1, you may optionally have your servers output debugging information. But there are no formal output requirements for servers to generate output either to the console, or to logfile(s).

Help Output

When GenericNode is run without parameters, produce a help message as below that identifies available client/server functionality. For example, if only TCP and UDP are implemented, then RMI messages are not shown.

```
GenericNode Usage:

Client:
uc/tc <address> <port> put <key> <msg>  UDP/TCP CLIENT: Put object into store
uc/tc <address> <port> get <key>        UDP/TCP CLIENT: Get object from store by key
uc/tc <address> <port> del <key>        UDP/TCP CLIENT: Delete object from store by key
uc/tc <address> <port> store            UDP/TCP CLIENT: Display object store
uc/tc <address> <port> exit             UDP/TCP CLIENT: Shutdown server
rmic <address> put <key> <msg>          RMI CLIENT: Put object into store
rmic <address> get <key>                RMI CLIENT: Get object from store by key
rmic <address> del <key>                RMI CLIENT: Delete object from store by key
rmic <address> store                    RMI CLIENT: Display object store
rmic <address> exit                     RMI CLIENT: Shutdown server

Server:
us/ts <port>            UDP/TCP: run udp or tcp server on <port>.
rmis                    Run the RMI Server.
tus <tcpport> <udpport> TCP-and-UDP SERVER: run TCP server on <tcpport> and UDP
                        server on <udpport> sharing the same key-value store.
alls <tcpport> <udpport> TCP, UDP, and RMI SERVER: run TCP server on <tcpport>, UDP
                        server on <udpport>, and RMI server sharing the same KV
                        store.
```

A nice feature to support in your GenericNode server jar is the ability to launch multiple types of servers (TCP, UDP, RMI) to operate at-the-same-time against the same back-end data store. This feature is nice, *not required*, and there is no extra credit available for implementing it, but it should be easy to support.

For this "tus" will launch a TCP and UDP server at the same time, and "alls" will launch a TCP, UDP, and RMI server all at the same time.

RMI registry

For distributed objects in Java, clients discover what remote objects are available by communicating with a central RMI registry server. The RMI registry allows servers to publish their list of hosted objects for discovery. RMI in Java uses a hybrid architecture where a centralized repository is used to track the available distributed objects, and which servers they reside on.

IMPORTANT:
For assignment #1, run only one instance of the RMI registry.
Once the RMI registry is started, a second instance cannot be started on the same network interface (i.e. eth0, network card). The RMI registry locks down a PORT which cannot be shared amongst multiple registry server instances.)

For this reason, check first if an RMI registry is running:

```
$ ps aux | grep rmi
wlloyd    2017  0.0  0.0  14224    976 pts/24   S+   16:46   0:00 grep --
color=auto rmi
wlloyd    3912  0.0  0.2 11480904 79920 pts/19 Sl   Oct21   0:17 rmiregistry -J-
Djava.class.path=GenericNode.jar
wlloyd   29988  0.1  0.1 685408 55120 ?        Sl   Oct21   2:04
/usr/lib/gnome-terminal/gnome-terminal-server

# Kill the existing RMI registry server, by specifying the PID of the
# existing server to terminate:
$ sudo kill 3912
```

When deploying new versions of the RMI server for testing, it is necessary to terminate and restart the RMI registry each time.

Testing the servers

Once the IP address of the server is discovered, point your client to this IP address and include the port number for TCP and UDP to support client/server interaction. RMI by default does not require a port to be specified.

Get, put, delete, store, and exit commands should be supported using each protocol developed.

Please follow as closely as possible the output format shown below.

Note that regardless of protocol the CLIENT output is essentially the same.

TCP CLIENT TO SERVER INTERACTION

The first parameter is "tc" for TCP client.
The second parameter is the server IP address.
The third parameter is the server port.

Replace localhost with your server IP address.
"1234" represents the service port. The client and server allow the port number to be specified.
Replace with the port used.

```
$ java -jar GenericNode.jar tc localhost 1234 put a 123
```
server response:put key=a

```
$ java -jar GenericNode.jar tc localhost 1234 put b 456
```
server response:put key=b

```
$ java -jar GenericNode.jar tc localhost 1234 get a
```
server response:get key=a get val=123

```
$ java -jar GenericNode.jar tc localhost 1234 del a
```
server response:delete key=a

```
$ java -jar GenericNode.jar tc localhost 1234 store
```
server response:
key:b:value:456:

```
$ java -jar GenericNode.jar tc localhost 1234 exit
```
<the server then exits>

UDP CLIENT TO SERVER INTERACTION

The first parameter is "uc" for UDP client.
The second parameter is the server IP address.
The third parameter is the server port.

UDP SERVERS LISTEN ON PORT A, SHOULD SEND ON PORT A+1
For example, if the UDP server listens on port 1234, it should respond on port 1235.

```
$ java -jar GenericNode.jar uc localhost 1234 put a 123
```
server response:put key=a

```
$ java -jar GenericNode.jar uc localhost 1234 put b 456
```
server response:put key=b

```
$ java -jar GenericNode.jar uc localhost 1234 get a
```
server response:get key=a get val=123

```
$ java -jar GenericNode.jar uc localhost 1234 del a
```
server response:delete key=a

```
$ java -jar GenericNode.jar uc localhost 1234 store
```
server response:
key:b:value:456:

```
$ java -jar GenericNode.jar uc localhost 1234 exit
```

RMI CLIENT TO SERVER INTERACTION

```
$ java -jar GenericNode.jar rmic localhost put a 123
server response:put key=a

$ java -jar GenericNode.jar rmic localhost put b 456
server response:put key=b

$ java -jar GenericNode.jar rmic localhost get a
server response:get key=a get val=123

$ java -jar GenericNode.jar rmic localhost del a
server response:delete key=a

$ java -jar GenericNode.jar rmic localhost store
```
server response:
key:b:value:456:

```
$ java -jar GenericNode.jar rmic localhost exit
```
Closing client...

RMI References

These RMI references may be helpful:
https://docs.oracle.com/javase/tutorial/rmi/index.html
https://docs.oracle.com/javase/tutorial/rmi/overview.html
https://docs.oracle.com/javase/tutorial/rmi/running.html
https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/
https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/codebase.html
https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/hello/hello-world.html
https://docs.oracle.com/en/java/javase/11/tools/remote-method-invocation-rmi-commands.html
https://docs.oracle.com/en/java/javase/11/docs/api/java.rmi/module-summary.html

TCP / UDP References

These may be helpful:
https://systembash.com/a-simple-java-tcp-server-and-tcp-client/
https://docs.oracle.com/javase/tutorial/networking/
https://docs.oracle.com/javase/tutorial/networking/sockets/index.html
https://docs.oracle.com/javase/tutorial/networking/datagrams/clientServer.html

<div style="border: 1px solid; padding: 10px; background-color: #d6e8ec;">

Project Submission –Java Maven Project Template

All Java submissions are **required** to provide a maven project build file (pom.xml). The build file should support two targets "clean" to purge binary files, and "verify" to build all project files. **To support Maven project submissions, a Java 11 LTS maven project template has been posted that includes a skeleton structure for implementing the project as a Maven project using Java 11.**

For Java submissions, use of the Maven template is strongly recommended. If the template is not used, groups should submit a functionally equivalent maven project that responds to the same targets: "clean" and "verify".

```
# to download the maven template project
wget http://faculty.washington.edu/wlloyd/courses/tcss558/assignments/a1/GenericNode.tar.gz

# extract maven template project
tar xzf GenericNode.tar.gz

# clean compilation files - should be performed before build
mvn clean

# compile maven project
mvn verify

# run maven project without any command line arguments - provides the help text
cd target
java -jar GenericNode.jar
```

For more information about maven, visit: **https://maven.apache.org/**

</div>

The recommended IDE for Java development is Apache Netbeans. The current release is Apache Netbeans 16. LTS releases have been discontinued in favor of a quarterly release cycle.

Download Apache Netbeans latest: **https://netbeans.apache.org/download/index.html**
Download Apache Netbeans 12 LTS: **https://netbeans.apache.org/download/nb120/nb120.html**
Download Apache Netbeans 11 LTS: **https://netbeans.apache.org/download/nb110/nb110.html**

Visual Studio Code can also build Java maven projects: **https://code.visualstudio.com/download**

No other Java IDE is recommended. If a working maven build file (pom.xml) is not provided, then detailed documentation is required that describes how to build the project executables from scratch. Failure to provide a project with a working maven build file (pom.xml) may result in the project being returned ungraded.

Testing Function and Performance

TCP, UDP, and RMI test scripts have been posted online at:

TCP:
http://faculty.washington.edu/wlloyd/courses/tcss558/assignments/a1/bigtest_tc.sh

UDP:
http://faculty.washington.edu/wlloyd/courses/tcss558/assignments/a1/bigtest_uc.sh

RMI:
http://faculty.washington.edu/wlloyd/courses/tcss558/assignments/a1/bigtest_rc.sh

These test files are only samples. To run these scripts, adjust the server and port BASH variables as needed to test your deployments.

You can check that your server has worked on the script by counting the number of resulting lines in the key value store at the conclusion of the test script as follows:

```
$java -jar GenericNode.jar rmic localhost store | wc -l

$java -jar GenericNode.jar tc localhost 1234 store | wc -l

$java -jar GenericNode.jar uc localhost 1234 store | wc -l
```

Assuming no blank lines, the count should be 70.

To obtain performance numbers of TCP, UDP, and/or RMI, run the scripts as follows:

```
#TCP
time ../bigtest_tc.sh > /dev/null

#UDP
time ../bigtest_uc.sh > /dev/null

#RMI
time ../bigtest_rc.sh > /dev/null
```

These are not the actual test files that will be used to grade projects. They are simple and basic. Please modify the test files to make more interesting and complex tests.

With your submission, please create a PDF file using Google Docs, MS Word, or another program. Include in the file performance numbers obtained using the above tests as follows:

Assignment 1 Performance Comparison TCP, UDP, RMI
TCP 17.513s
UDP 53.112s
RMI 14.843s

Note the example times here are not actual performance times. These are random numbers.

Teams (optional)
It is strongly encouraged to complete this assignment in three person teams. One and two person teams are acceptable as well.

If choosing to work in teams, *only one* person should submit the team's tar gzip project source archive file and the performance report PDF file to Canvas.

Additionally, *EACH* member of a team should submit a confidential **effort report** describing team member participation. **Effort reports** are submitted INDEPENDENTLY and in confidence (i.e. not shared) by each team member.

<u>Effort reports are not used to directly numerically weight assignment grades.</u>

Effort reports should be submitted as a PDF file named: "effort_report.pdf". Use Google Docs, MS Word, or Open/Libre Office to create and export a document in PDF format.

For assignment 0, the effort report should consist of a one-third to one-half page narrative description describing how the team members worked together to complete the assignment.

Effort Report Narrative - address the following points:

1. Describe the key contributions made by each team member.
2. Describe how working together was beneficial for completing the assignment. This may include how the learning objectives of using EC2, Docker, Docker-machine, and haproxy were supported by the team effort.
3. Comment on disadvantages and/or challenges for working together on the assignment. This could be anything from group dynamics, to commute challenges, to faulty technology.
4. At the bottom of the write-up provide an effort ranking from 0 to 100 for each team member. Distribute a total of 100 points among both team members. Identify team members using first and last name. For example:

        John Doe
        Research   25
        Design   40
        Coding   30
        Testing   45

        On Li
        Research   40
        Design   35
        Coding   40
        Testing   25

        Jane Smith
        Research   35
        Design   25
        Coding   30
        Testing   30

Please do not simply put equal scores for all team members. It is _very unlikely_ every team member contributed exactly the same effort on all aspects. Please provide <u>realistic</u> rankings.

Team members may not share their **effort reports**, but should submit them independently in Canvas as a PDF file. Failure of team members to submit the **effort reports** will result in the assignment receiving NO GRADE…

<u>Disclaimer regarding pair programming:</u>
The purpose of TCSS 558 is for everyone to gain experience developing and working with distributed systems and requisite compute infrastructure. Group/pair programming is provided as an opportunity to harness teamwork to tackle programming challenges. But this does not mean that teams consist of one champion programmer, and a other observers that simply watch the champion! The tasks and challenges should be shared as equally as possible.

What to Submit

To submit the assignment, teams should build a single tar gzip archive file that contains **all** project source code in a main project directory. This could be the Netbeans project folder. In the project directory there should be two directories for the Docker containers: docker_server and docker_client. The folders must be updated to include your GenericNode.jar file and they must support building a functioning TCP, UDP, and/or RMI server container.

PDF files with performance results should be submitted as a separate file in Canvas.

Grading Rubric

This assignment will be scored out of 100 points, while as many as 123 points are possible.

Functionality	63 points
5 points	TCP client/server put
5 points	TCP client/server get
5 points	TCP client/server del
5 points	TCP client/server store
1 point	TCP client/server exit
5 points	UDP client/server put
5 points	UDP client/server get
5 points	UDP client/server del
5 points	UDP client/server store
1 point	UDP client/server exit
5 points	RMI client/server put
5 points	RMI client/server get
5 points	RMI client/server del
5 points	RMI client/server store
1 point	RMI client/server exit

Miscellaneous	60 points
10 points	Use of multiple server threads
10 points	Key-value store synchronization of concurrent writes
10 points	Performance comparison using at least two protocols
10 points	Working docker containers
10 points	Client/server interaction (i.e. output) matches specification
5 points	Coding style, code reuse among clients and servers
5 points	Program compiles, instructions provided or Netbeans used

List of Common Docker Commands

# Docker build
sudo docker build -t tcss558server .
sudo docker build -t tcss558client .

# Run docker container in the background
sudo docker run -d --rm tcss558server
sudo docker run -d --rm tcss558client

# Docker "shell" to a container
sudo docker exec -it <container-id> bash

To display all containers running on a given docker node:
```
docker ps -a
```

To stop a container:
```
docker stop <container-id>
```
For example:
```
docker stop cd5a89bb7a98
```

Also `docker kill` will kill a running container and `docker rm` will remove a container which has exited but is no longer running.

Document History:
v.10   Initial version