Assignment 1

Version 0.10

Key Value Store – Client/Server

Due Date: Friday November 10th, 2017 @ 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, writes should be synchronized.
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 then be expanded to implement a client/server using one or both of the protocols: UDP (connection-less), and remote objects (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 RDP.

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 23% extra credit**.

The preferred implementation for assignment #1 is in Java 8. Students are free to implement assignment #1 in C, C++, or Python if preferred. **Solutions in alternate languages must include documentation to describe how to operate the client and server in the alternate language.** All operations including setup must be explained. Components must be deployed using server and client docker containers. If the operation of any client or server functions (put, get, del, store, exit) is unclear, no credit will be granted for these operations.

Docker for Assignment #1

All solutions must include a Server Dockerfile and a Client Dockerfile to support creating server and client containers. Clients and servers must be able to communicate on the local subnetwork shared among containers of a single Docker host. The client container will use the docker container private network IP address to facilitate communication with the server.

To support working with Docker containers, Dockerfiles for the client and server have been provided and can be downloaded here: (*feel free to use these, or develop new Dockerfiles...*)

http://faculty.washington.edu/wlloyd/courses/tcss558/assignments/a1/a1_dockerfiles.tar.gz

To extract a tar gzip file use the command: (x for extract, z for unzip, f for file)

tar xzf a1_dockerfiles.tar.gz

Then cd into the individual docker_server or docker_client directories to build the docker images.

The sample dockerfiles includes a placeholder GenericNode.jar Java class archive file. For assignment #1, you're to develop the GenericNode.jar which implements clients and servers for TCP, UDP, and/or RMI.

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

When building your docker_server container, you should 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, but then test the 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 my GenericNode is run without parameters, I also produce the following help is produced. It is optional to include the help message, but nice to do:

```
Usage:
Client:
uc/tc <address> <port> put <key> <msg>  UDP/TCP CLIENT: Put an object into store
uc/tc <address> <port> get <key>  UDP/TCP CLIENT: Get an object from store by key
uc/tc <address> <port> del <key>  UDP/TCP CLIENT: Delete an 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 an object into store
rmic <address> get <key>  RMI CLIENT: Get an object from store by key
rmic <address> del <key>  RMI CLIENT: Delete an 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/TCP-and-UDP SERVER: run server on <port>.
tus <tcpport> <udpport>  TCP-and-UDP SERVER: run servers on <tcpport> and <udpport> sharing same key-value store.
alls <tcpport> <udpport>  TCP, UDP, and RMI SERVER: run servers on <tcpport> and <udpport> sharing same key-value store.
rmic  RMI Server.
```

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

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 follows a hybrid architecture using a centralized repository with distributed object servers.

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 can not 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
```

If you see rmi registry servers, you will want to KILL and restart them whenever restarting (i.e. redeploying) your RMI server:

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
<the server then exits>

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/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/javase/tutorial/rmi/running.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/sockets/index.html
https://docs.oracle.com/javase/tutorial/networking/datagrams/clientServer.html

Suggested Integrated Development Environment (IDE) / Project Build Files

For maximum potential for ***partial credit***, students may submit their project as a Netbeans project created with the Netbeans IDE, Oracle's Java IDE. By providing projects as a Netbeans project, it will be possible for the grader to build your source and fix potential issues to support partial credit. For example, a student may nearly have the code correct, but a small detail prevents operation. If the grader can rapidly fix the code, a lot of partial credit may be awarded.

Download Netbeans 8.2 here:
https://netbeans.org/downloads/

Alternatively, students not using Netbeans may submit all requisite project build files, as well as descriptive documentation which clearly states how code can be rebuilt. If the grader is easily able to rebuild your projects, then there is higher potential for partial credit.

If no build files are provided, and/or no subsequent documentation describing how to build your projects, **it will not be possible to issue partial credit** for functionality that is nearly complete with minor bugs. In this case, only by reading the code will the grader attempt to issue partial credit if portions of the program do not work correctly.

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

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
```

With your submission, please create a PDF file created with Google Docs. 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 bogus.

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
10 points	Performance comparison using at least two protocols
10 points	Working docker containers
10 points	Client/server interaction matches specification
5 points	Coding style, code reuse among clients and servers
5 points	Program compiles, instructions provided or Netbeans used

Teams (optional)

O*ptionally*, this programming assignment can be completed with two person teams.

If choosing to work in pairs, ***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 an **effort report** on team participation. **Effort reports** are submitted INDEPENDENTLY and in confidence (i.e. not shared) by each team member.

Effort reports are not used to directly numerically weight assignment grades.

Effort reports should be submitted as a PDF file named: "effort_report.pdf". Google Docs and recent versions of MS Word provide the ability to save or 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. The description should include the following:

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 65
    Design 42
    Coding 30
    Testing 80

    Jane Smith
    Research 35
    Design 58
    Coding 70
    Testing 20

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

Disclaimer regarding pair programming:
The purpose of TCSS 558 is for everyone to gain experience developing and working with distributed systems and requisite compute infrastructure. 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 second observer simply watching the champion! The tasks and challenges should be shared as equally as possible.

Helpful Hints

# 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