CSCU9V5: Concurrent & Distributed Systems

Distributed Laboratory 2 - RMI

This lab contains a checkpoint. All code you present must be your own work and you will discuss with the demonstrator the details of it.

The main aim of this lab is to have a clear idea of how RMI can be build and how it runs.

Taking the RMI code presented in the lecture as a base[1], write a *client/server system.* The client side consists of both *producer thread* and a *consumer thread*. Both threads access a *remote message queue* object on the server side, using its methods. Initially, have the server and client side on the same host.

Before start coding, read in detail the provided files. Start from MessageQueueImp.java, then the Producer.java and the Consumer.java. Compare this given implementation with the general principles in the lecture slides, and check how each single step has been implemented in the code.

In the following, compiling and running from the command line might help following the various steps.

Then, the simplest way to run the programs is

1. **compile** all files;
2. generate **stub** and **skeleton** for the remote object*:
   rmic MessageQueueImpl
3. start the **name registry** * :
   rmiregistry
4. **run** the **remote object** and the **factory**, which will start the producer and consumer ** ;

* You can use **cmd** (as discussed in the lecture notes) to run

rmic MessageQueueImpl
rmiregistry

Make sure you do so from the directory where you have your class files.

** When you run the programs please ensure that you use the java switch

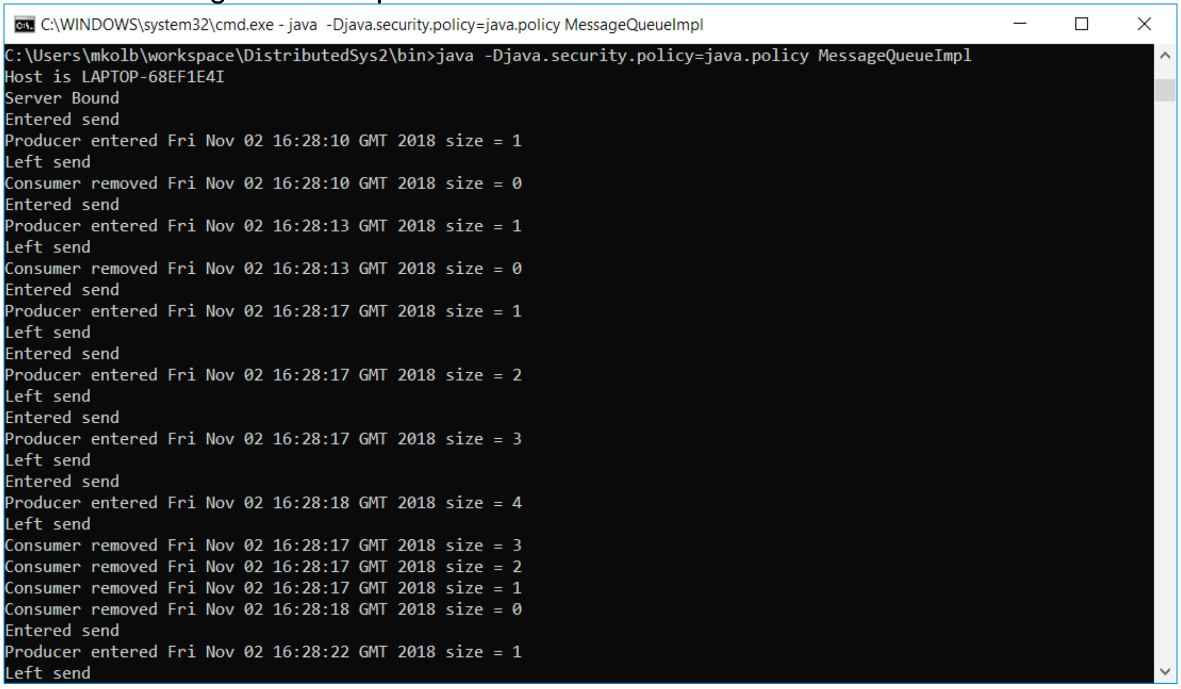
-Djava.security.policy=java.policy.

Also, the java.policy file must be in the directory where your source files are.

[1] Chapter 15 of Silberschatz also helps, as does your second laboratory on <u>concurrent</u> systems.

The following pictures show a possible run of the system.

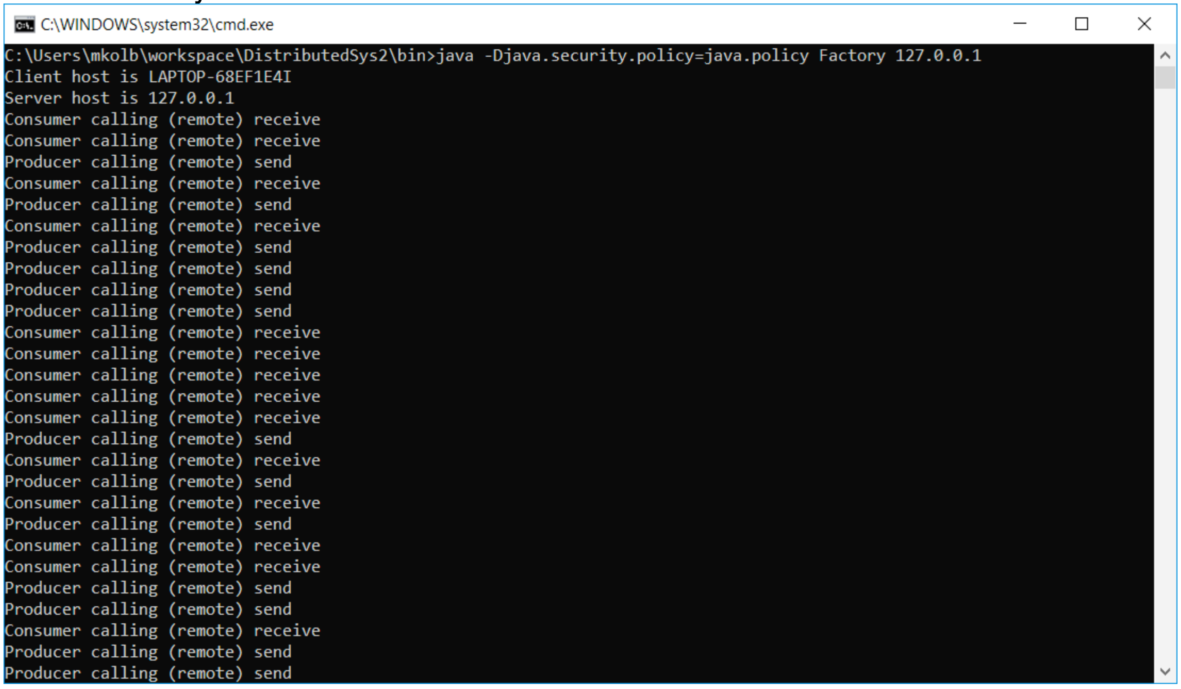
Server: MessageQueueImpl



C:\WINDOWS\system32\cmd.exe - java -Djava.security.policy=java.policy MessageQueueImpl

```
C:\Users\mkolb\workspace\DistributedSys2\bin>java -Djava.security.policy=java.policy MessageQueueImpl
Host is LAPTOP-68EF1E4I
Server Bound
Entered send
Producer entered Fri Nov 02 16:28:10 GMT 2018 size = 1
Left send
Consumer removed Fri Nov 02 16:28:10 GMT 2018 size = 0
Entered send
Producer entered Fri Nov 02 16:28:13 GMT 2018 size = 1
Left send
Consumer removed Fri Nov 02 16:28:13 GMT 2018 size = 0
Entered send
Producer entered Fri Nov 02 16:28:17 GMT 2018 size = 1
Left send
Entered send
Producer entered Fri Nov 02 16:28:17 GMT 2018 size = 2
Left send
Entered send
Producer entered Fri Nov 02 16:28:17 GMT 2018 size = 3
Left send
Entered send
Producer entered Fri Nov 02 16:28:18 GMT 2018 size = 4
Left send
Consumer removed Fri Nov 02 16:28:17 GMT 2018 size = 3
Consumer removed Fri Nov 02 16:28:17 GMT 2018 size = 2
Consumer removed Fri Nov 02 16:28:17 GMT 2018 size = 1
Consumer removed Fri Nov 02 16:28:18 GMT 2018 size = 0
Entered send
Producer entered Fri Nov 02 16:28:22 GMT 2018 size = 1
Left send
```

Client: Factory class to start Producer and Consumer Threads:



C:\WINDOWS\system32\cmd.exe

```
C:\Users\mkolb\workspace\DistributedSys2\bin>java -Djava.security.policy=java.policy Factory 127.0.0.1
Client host is LAPTOP-68EF1E4I
Server host is 127.0.0.1
Consumer calling (remote) receive
Consumer calling (remote) receive
Producer calling (remote) send
Consumer calling (remote) receive
Producer calling (remote) send
Consumer calling (remote) receive
Producer calling (remote) send
Producer calling (remote) send
Producer calling (remote) send
Producer calling (remote) send
Consumer calling (remote) receive
Consumer calling (remote) receive
Consumer calling (remote) receive
Consumer calling (remote) receive
Consumer calling (remote) receive
Producer calling (remote) send
Consumer calling (remote) receive
Producer calling (remote) send
Consumer calling (remote) receive
Producer calling (remote) send
Consumer calling (remote) receive
Consumer calling (remote) receive
Producer calling (remote) send
Producer calling (remote) send
Consumer calling (remote) receive
Producer calling (remote) send
Producer calling (remote) send
```

Try with more than 1 client running at once. Use a delay, as standard, and also use **println()**s to illustrate what is happening, as needed.

Q1. Does the dynamic behaviour change if the methods of the message queue are not sychronised? How? Why?

Next, we want to have the server and client on separate hosts. Again, you will have more than one client operating at once.

You may find **java.InetAddress.getLocalHost()** useful with **System.out.println (..)** in **both** the server and client to follow what is happening. One example is in the the client side (Factory):

```
public static void main(String argv[]) {
 if ((argv.length < 1) || (argv.length > 1))
   { System.out.println("Usage: [remote-host-name]") ;
     System.exit (1) ;
   }

 String server_host = argv[0] ;
 …

 Naming.lookup("rmi://"+server_host+"/MessageQueue");
```

This allows you to enter the server host name when you run your client.

Checkpoint.

Finally **try your client with someone else's server**. Although outside of the checkpoint, this part of the lab is very important for future developments.