**Name:Bhagyashree Narayan Shinde**

**Student Id:1001552353**

**Assignment No.2**

**AIM: -** To develop **Online Advising Simulation using RPC & Message Queuing**

**OBJECTIVE:**

1. Exposure to application development in distributed environment

2. Understanding two communication models for distributed processes

a.) Remote Procedure Call / Remote Method Invocation (Synchronous Communication)

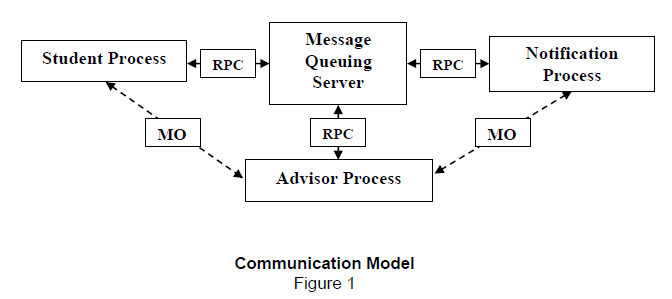
b.) Message Oriented Communication (Asynchronous persistent Communication)

3. Understanding the message queuing systems.

**Project Specification:**

* Integrated Development Environment used :-Eclipse Europa
* Language used for coding application:-Java.
* Java Development Kit used: JDK1.7
* Tool used for building GUI :Java Swing GUI widget toolkit.

In this lab you will simulate an online advising system which is similar to what you might have used for registering your classes. Students request clearance for a course from the advisor. At some other time the advisor approves or disapproves the request and then the student is notified of the advisor’s decision. You will require 4 processes to simulate this: a student process, an advisor process, a notification process and a message queuing server (MQS). The student process, advisor process and notification process communicate through the message queuing server (message oriented middleware). Communication from these processes to the message queuing server is through remote procedure calls. Please see figure 1:



Note: Student, Advisor and Notification Process do not communicate directly. The dotted arrow in figure 1 implies the logical flow of messages. Also, “RPC” is interchangeable with “RMI,” depending on your selected programming language.

*Student Process:* The Student process works on behalf of the student. It takes as input (from the command prompt) the name of the student and the course for which the student wants clearance. The Student process should then send a message through the MQS to the Advisor seeking clearance. It should keep doing this (under control of the user) until it is killed.

*Advisor Process:*

When it is started, the Advisor Process should contact the MQS for any message from the Student process. If there are any messages then it should retrieve them and approve or disapprove the requests based on a random probability. It should then send a message for each request (message) through the MQS to the Notification process so that it (the Notification process) can notify the Student process about the decision. If there are no messages for the Advisor process in the MQS then it should sleep for 3 seconds and again contact back to check for messages. It should keep doing this until is killed. You should also print out a message for each request (message) that the Advisor process processes (whether it is approved or not).

*Notification Process:*

The Notification process notifies the Student process about the Advisor’s decision. To simulate this you just have to print on the command prompt the student name, course and the advisor decision. The Notification process should check MQS for messages from the Advisor process. If there are any messages then it should retrieve them and notify the student by printing as stated above. If there are no messages then it should sleep for 7 seconds and then contact the MQS again. It should keep doing this until is killed.

*Message Queuing Server (MQS):*

This process simulates the Message Queuing server. MQS should have the following:

1. It should be able to store or return a message from or to any requesting process.

2. You should have only one physical queue (single array, linked list or queue, etc.) to store the messages from both Student and Advisor process.

3. Messages once retrieved should be physically deleted from the queue.

4. Messages should be persistent, i.e. even if you shut down the server the messages should not be lost. For that you should store the messages in a file. When the MQS is restarted it should reload all the messages from the file to your main memory data structure (array or linked-list etc). We will assume that the MQS will be killed only after all the 3 other processes are killed so that you don’t need to worry about problems that would arise if it were killed while in communication.

*Other Specifications:*

1. The MQS will be started first. The other 3 processes can be started or stopped in any order. So make sure that your application works fine independently of the order in which these 3 processes are started and stopped.

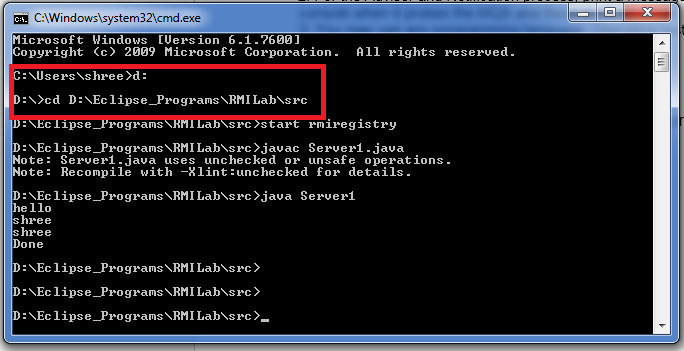
2. For the Advisor and Notification process, print a message (like “no message found”) on the console when it probes the MQS and there is no message.

3. You may use any programming language, Operating system, or Server to complete this lab.

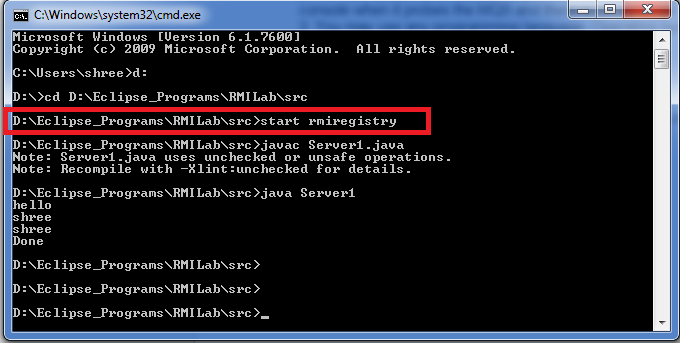
Steps to be followed for execution of application:

**NOTE:**Import the project in eclipse after extracting the project from zip folder.

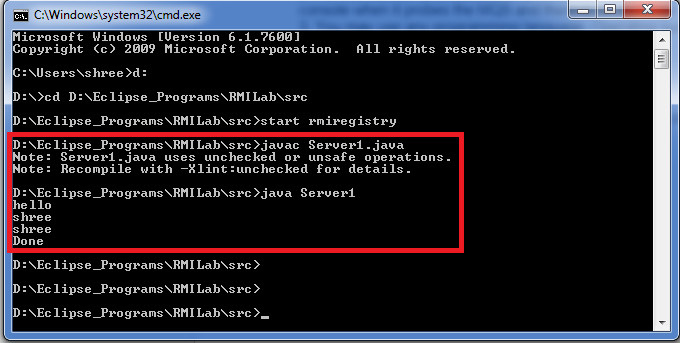
1)Initially start with cmd by chaning directory to path or place where project resides



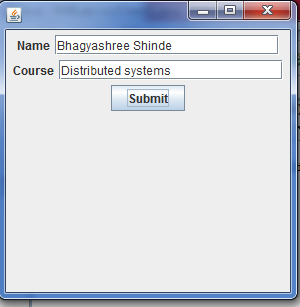
2)start rmiregistry:



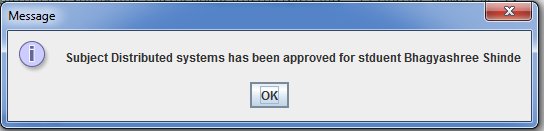
3)Compile Message Queueing Server and run :



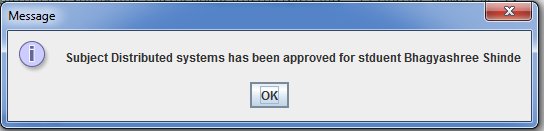
4) Now execute student process i.e client1.java in my case/project from eclipse and enter student details:



5)After entering student details run advisor process i.e Advisor.java:

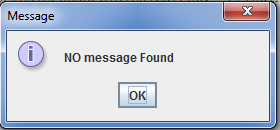


6)After successful execution of advisor process run notification process i.e Notification.java:



Note: Moreover, if server goes down before processing client request, the client data is stored in text file named=”thetextfile” .whenever server is up, it checks file for data or any incomplete request that’s yet to be processed.

If there are no student details entered or no student details found advisor and notification process replies with “No message found”.



**INPUT :** Student details : Student Name – name of the student requesting advisor decision

Course Name – The course for which student wants approval.

**OUTPUT:** Advisor decision based on student request notified by notification process.

**Conclusion:** This project exposes to distributed application and understand two communication models for distributed processes

a.) Remote Procedure Call / Remote Method Invocation (Synchronous Communication)

b.) Message Oriented Communication (Asynchronous persistent Communication)

Moreover it also helps to understand the message queuing systems.

**References:**

**Below are the links referred for developing this project**

1. <https://docs.oracle.com/javase/tutorial/rmi/index.html/>

1. <https://stackoverflow.com/questions/>