Northeastern University

CS 6650 Scalable Dist Systems

Homework Set #2 [100 points]

Name: Nisarg Patel
Email: patel.nisargs@northeastern.edu

<u>INSTRUCTIONS:</u> Please provide clear explanations I your own sentences, directly answering the question, demonstrating your understanding of the question and its solution, in depth, with sufficient detail. Submit your solutions [PDF preferred]. Include your full name. Do not email the solutions.

I. **Java Threads – Single and Multi-threading implementation** [20 points]

This small programming project is preparation toward the next set of programming assignments.
It focuses on implementing a simple Java server using single threading first and multithreading later.
Study the necessary sections from Buyya Ch. 14
http://www.buyya.com/java/Chapter14.pdf

Implement simple client – server communication to demonstrate:

a) Single threading
b) Multi-threading

Ans:

Study **Chapter 4 and 5, Chapters 6 and 9 from** Coulouris Book to answer the below:

II. Answer the following questions using explanation and diagrams as needed. No implementation needed.

1. 4.2 [5 points]

A server creates a port that it uses to receive requests from clients. Discuss the design issues concerning the relationship between the name of this port and the names used by clients.

Ans:

If the server shares the port number and IP address to the clients, then the location transparency is not achieved by the system. The server is bound by the host and the port, thus have to run on the same machine all the time. The location transparency can be achieved by using a binder (name server), that can translate to the location of server during runtime. The client would refer to the server name which would be different from the port name. But this would mean that mobility transparency is still not achieved as client is still bound by the server it is connected.

2. 4.15 [10 points]

Ans:

Since we are assuming that messages follow the same order as they are sent by the sender, w can use this information to create a simple response-based scheme to retransmit the undelivered datagrams. To allow multiple senders to send messages, a global counter can be placed that each sender will have access to. The global counter would keep track of the last message sent by any of the sender. Then the message can include the counter in it. The recipient can then keep track of the datagrams that they received, by checking the value of the counter. Since there are only a few messages that are dropped, the recipient can know which counter numbered messages are dropped, and thus recipient can request message retransmission for those messages. This would require sender to keep track of recent messages sent. Since client would only send a message to request for undelivered messages, the recipients are not sending messages withing a particular time limit. But this would lead to sender to keep track of all the messages that they sent. One way to overcome this problem is that each sender can keep a maximum number of latest messages to be stored which could be much larger than the probability of a single message to be undelivered. One more way to solve this is by extracting information about the delivered messages from the message send by recipient. Thus removing the message from buffer if it is delivered to all the recipients. Recipients can even send some acknowledgements after a specified number of messages received.

3. 5.11 [5 points]
Ans:
Since the *vote* method is to be executed on server, with no return expected, both the parameters: **candidateName** and **voterNumber** are input parameters.

The parameter type for the *result* method depends on the functionality of the method:

- If *result* method should return the **candidateName** and **numberOfVotes** of the winner, then both the parameters are output parameters.
- If *result* method should return the **numberOfVotes** of the **candidateName** provided (that is to get the number of votes for a particular candidate)**,** then candidateName is the input parameter and **numberOfVotes** is the output parameter.

4. 5.13 [5 points]
Ans: Using the first functionality of result method discussed in the previous question.

CORBA IDL:
```
// In file Candidate.idl
struct Candidate {
 string name;
 long votes;
};

interface VoteService {
 void vote(in string candidateName, in long voterNumber);
 void result(out Candidate winner);
};
```

JAVA:
```
import java.rmi.*;

class Candidate {
 String name;
 int votes;
};

public interface VoteService extends Remote {
  void vote(String candidateName, int voterNumber) throws RemoteException;
  Candidate result() throws RemoteException;
};
```

In CORBA IDL, as shown in the code, the input and output parameters is specified with the parameters itself with keywords: in and out respectively. Whereas in JAVA, the input parameters are passed as arguments and output parameters is the return type of the methods.

5. 5.22 [10 points]

Ans:

i) For Single Threaded 2 requests:

Time for 1 request = client compute time + marshalling time + client OS send processing time + network time + server OS receive process time + unmarshalling time + server request processing time + marshalling time + server OS send process time + network time + client OS receive processing time + unmarshalling time

= 5ms + 0.5ms + 0.5ms + 3ms + 0.5ms + 0.5ms + 10ms + 0.5ms + 0.5ms + 3ms + 0.5ms + 0.5ms

= 25 ms

Thus total time for 2 request for single threaded server = 2*25ms **= 50ms**

ii) For 2 concurrent requests on single processor:

The second request on client side will start executing as soon as first request gets sent. The same is applied for server as well. Thus the total time to process 2 requests is

= client compute time + marshalling time + client OS send processing time + network time + server OS receive process time + unmarshalling time + server request processing time + marshalling time + server OS send process time + server OS receive process time + unmarshalling time + server request processing time + marshalling time + server OS send process time + network time + client OS receive processing time + unmarshalling time

= 5ms + 0.5ms + 0.5ms + 3ms + 0.5ms + 0.5ms + 10ms + 0.5ms + 0.5ms + 0.5ms + 0.5ms + 10ms + 0.5ms + 0.5ms + 3ms + 0.5ms + 0.5ms

= 37ms

The need for asynchronous invocation depends on whether the method can create race conditions. If not, (for example just getting values from a method), the invocation can be made asynchronous.

III. Answer the following questions using explanation and diagrams as needed. No implementation needed.

6. 6.8 [5 points]

Ans:

We use the publish – subscribe model to design the notification mailbox system. We follow the basic channel-based notification scheme for this mailbox service. So, subscribers can subscribe to a particular channel and receive all events based on that channel. For this subscribe(String channel) method can be used. Since subscribers can specify when they require notifications, it can be designed in either of 2 ways. First, the subscriber can call setNotificationTime(Time timeInterval) to get the notifications after specific time intervals. Second, the subscriber can also call setShouldGetNotication(boolean true/false) to let subscriber get or not get notifications from this channel till this boolean value is changed. The second method also helps inactive users to pause the notification mailbox service for that channel for some time. These variables will also be stored remotely so that whenever the subscribers crash, they can restore back from the same point.

7. 6.14 [5 points]

Ans:

To help achieve the functionality to receive alarms only from a given location, we can make use of a message consumer which is used to receive messages from a specific topic with applying filters. These filters know as message selector is a filter defined on the message's header values and properties parts. Firstly, we can add location in the properties part of the message in FireAlarmJMS. Then, we can use message selector on properties part and filter on Location on extended FireAlarmConsumerJMS.

8. 6.15 [5 points]

Ans:

Distributed Shared Memory is used primarily in development of parallel applications or in distributed systems where the shared data items need to be directly accessible.
It is not generally preferred where clients have the server data in the form of abstract data and clients access the remote data by requesting to the server. This is when the distributed applications need to be modular and for security reasons. Also for security reasons, we would require a shared region for each client and thus increasing the costs.

9. 9.1 [5 points]

Ans:

The request-reply protocol is synchronous. Whereas Simple Object Access Protocol is designed to allow asynchronous communication of client and server over the internet. The request-reply has its own message structure and uses TCP where the SOAP is based on XML and can be used on various transport protocols, but it is most popular on HTTP.

Use of asynchronous messages by SOAP is preferrable over the Internet because SOAP uses HTTP POST method for client request and its response as a reply message. The SOAP combines usage of XML and HTTP which are considered standard protocol over the Internet. Furthermore, SOAP messages can be passed from computer to computer till the destination server. These intermediate stops can be useful for different complex middleware services such as security and/or transactions.

10. 9.8 [5 points]

Ans:

The servlet container is important in deployment of a web service. The servlet is the running process of a service. These servlets are run inside a servlet container. Thus, the servlet container loads, initializes and executes different servets. The servlet containers are also responsible for mapping the client requests into a specific skeleton code. It then translates into Java code while passing the request to the respective method present in the servlet. When the reply is generated by that method, the same skeleton is used to translate the reply back into SOAP format.

IV. See the Coding Tutorial PDF provided, and the below references. Do your own study of RMI Java examples to implement this. Implement a Java RMI Application in which the Client object is sending a list of 10 integers to the Server, and a remote method ['sort()'] o the server returns a sorted version of the same list back to the Client. [20 points]

Ans:

References

Java RMI
https://www.cs.uic.edu/~troy/fall04/cs441/rmi/calc/index.html