Important Instructions:

1) **Open this MS-Word document and start writing answers below each respective question given on page 2.**
2) **Answers the question in the same sequence in which they appear.**
3) **Provide to the point and concrete answers.**
4) **First read the questions and understand what is required of you before writing the answer.**
5) **Attempt the paper yourself and do not copy from your friends or the Internet. Students with exactly similar answers or copy paste from the Internet will not get any marks for their assignment.**
6) **You can contact me for help if you have any doubt in the above instructions or the assignment questions.**
7) **All questions must be attempted.**
8) **Do not forget to write your name, university ID, class and section information.**
9) **Rename you answer file with your university ID# before uploading to SIC.**
10) **When you are finished with writing your answers and are ready to submit your answer, convert it to PDF (no MS Word) and upload it to SIC unzipped, before the deadline mentioned on SIC.**
11) **Do not make any changes to the format provided.**
12) **Failure in following the above instructions might result in deduction of marks.**

Semester Assignment
Course: - Distributed Computing

Deadline: - Mentioned on SIC **Marks: - 20**

Program: - MS (CS) **Dated: 17 September 2020**

Student Name: _AWAID ULLAH_ Student ID#:__12714__

Class and Section:___MS(CS)__

<u>Question:</u> Assume you have a Client Server Environment in which the client request the server to multiply three given number i.e 67, 90, 34, and return the result. Discuss the steps of the system in each of the following scenarios.

<u>a)</u> How the Request-Reply Protocols functions will be used with UDP (refer to figure 5.3 in book), how will be the message identifiers used, what will be its failure model, how time outs will be used, how will the system handle duplicate messages and how will the system react if reply is lost. (8)

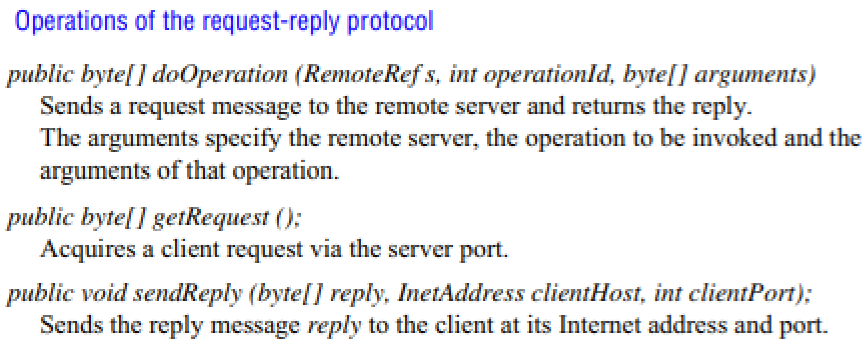
<u>b)</u> How can the above system implemented using Remote Procedure Calls (RPC)? (Hint: Read Section 5.3.2 in the book). (6)

<u>c)</u> How can the above system implemented using Remote Method Invocation (RMI)? (Hint: Read Section 5.4.2 in the book). (6)

<u>Question:</u> Assume you have a Client Server Environment in which the client request the server to multiply three given number i.e 67, 90, 34, and return the result. Discuss the steps of the system in each of the following scenarios.

<u>A)</u> How the Request-Reply Protocols functions will be used with UDP (refer to figure 5.3 in book), how will be the message identifiers used, what will be its failure model, how time outs will be used, how will the system handle duplicate messages and how will the system react if reply is lost

The protocol we describe here is based on three communication primitives, operates, gets a request and sends a response (as shown) to perform the operation. The request-response protocol matches the request with the response. It can be designed to provide certain delivery guarantees. If UDP datagrams are used, delivery guarantees must be provided by the request-response protocol, which can use the server response message as a confirmation of the client request message. The figure shows three communication primitives.
The client uses the do Operation method to call remote operations. Its parameters specify the remote server and the operation to be invoked, as well as any other information (parameters) required for the operation. The result is a byte array containing the response. Suppose the client that called do Operation collected


Operations of the request-reply protocol

public byte[] doOperation (RemoteRef s, int operationId, byte[] arguments)
Sends a request message to the remote server and returns the reply.
The arguments specify the remote server, the operation to be invoked and the arguments of that operation.

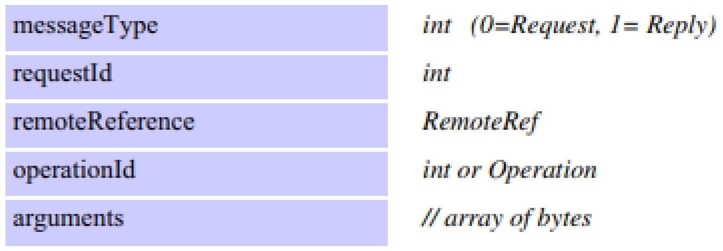
public byte[] getRequest ();
Acquires a client request via the server port.

public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);
Sends the reply message *reply* to the client at its Internet address and port.

The parameters in the byte array and the result of dividing the returned byte array. The first parameter of do Operation is an instance of the Remote Ref class, which represents a reference to a remote server. This class provides a method for obtaining the Internet address and port of the associated server. The do Operation method sends a request message to the server. The Internet address and port of the server are specified in the remote reference provided as a parameter. After sending the request message, perform the operation call receive to get the response message, and then extract the result from it and send it back to the caller. The caller of the do operation is not blocked until the server performs the requested operation and sends the response message to the client process.

The server process uses get Request to obtain service requests, as shown in the figure. Public do Operation byte request-response protocol operation (remote reference, in operation ID, byte parameter) sends a request message to the remote server and returns a response. The parameters specify the remote server, the operation to be invoked, and the parameters of the operation. Public byte get Request; Get the client request through the server port. Public invalid send reply (byte response, the client host has an Internet address, in the client port); the response of the response message is sent to the client's Internet address and port. . When the server invokes the specified operation, it will use send Reply to send the response message to the client. When the client receives the response message, the original do operation will be unblocked, and the execution of the client program will continue.

The second field "Request ID" contains the message identifier. The do operation in the client generates a request ID for each request message, and then the server copies these IDs to the corresponding response message. This allows do Operation to verify that the response message is the result of the current request and not the result of a previous delayed call. The third field is the remote reference. The fourth field is the identifier of the operation to be invoked. For example, the operations in the interface can be numbered 1, 2, and 3. If the client and server use a common language that supports reflection, the representation of the operation itself can be placed in this field.



messageType	*int (0=Request, 1= Reply)*
requestId	*int*
remoteReference	*RemoteRef*
operationId	*int or Operation*
arguments	*// array of bytes*

Message identifiers:

Any scheme that involves managing messages to provide other attributes (such as reliable messaging or request-response communication) requires that each message have a unique message identifier through which it can be referenced. The message identifier consists of two parts:

1. Request an identifier, which is extracted from an increasing sequence of integers through the sending process;
2. The identifier of the sender process, such as its port and Internet address.

The first part makes the identifier unique to the sender, while the second part makes the identifier unique in a distributed system. (The second part can be obtained independently-for example, from the received message if UDP is used.)
When the value of the request ID reaches the maximum value of an unsigned integer (for example, 232-1), it will be reset to zero. The only restriction here is that the lifetime of the message identifier must be much less than the time required to exhaust the values in the integer sequence.

Failure model:

If these three primitives implement Operation, get Request and send Reply on the UDP datagram, they will suffer the same communication failure.

• They suffered failure due to negligence.
• There is no guarantee that mail will be sent in the order of the sender.

In addition, the protocol may experience process failures (see section). We assume that the process has crashed and failed. In other words, when they stopped, they remained stopped-they did not produce Byzantine behavior.
To consider the server failure or abandon the request or response message, perform an "action" to receive the response message from the server using a timeout. The measures taken in the event of exceeding the deadline depend on the delivery guarantee provided.

Timeouts:

There are different options regarding the operations that do Operation can perform after the delay. The simplest option is to immediately return from do Operation and indicate to the client that do Operation failed. This is not the usual method-the timeout may be caused by the loss of the request or response message, in which case the operation will be performed. In order to make up for the possibility of missing messages, do Operation repeatedly sends request messages until a response is obtained or it is reasonably determined that the delay is due to the lack of response from the server rather than the missing message. Finally, when do Operation returns, it will abnormally notify the client that no results have been received.

Duplicate request messages:

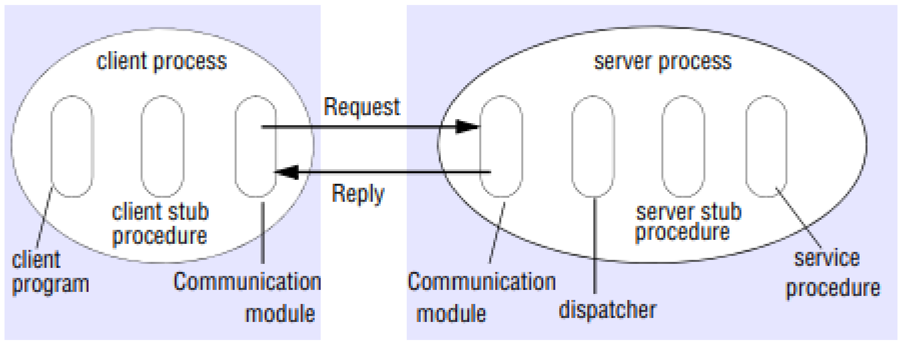
In the case of resending the request message, the server may receive the request message multiple times. For example, the server may receive the first request message, but it will take longer than the client timeout to complete the command and return a response. This may cause the server to perform more than one operation on the same request. To avoid this, the protocol aims to identify consecutive messages (from the same client) with the same request ID and filter out duplicates. If the server has not sent a response, it does not need to take any special action-it will forward the response after the operation is completed.

Reply lost

If the server has already sent a response when it receives a repeated request, it will need to rerun the operation to get the result unless the result of the original execution is stored. Some servers may run their operations multiple times and get the same results each time. Idempotent is an operation that can be performed multiple times, and its effect is the same as performing it only once. For example, the operation of adding an element to a set is an idempotent operation, because every time the operation is performed, it will have the same effect on the set, while the operation of adding an element to a sequence will not be an idempotent operation because it will The sequence is expanded during the second run. The operations of the server are idempotent, and no special measures are required to avoid performing its operations multiple times.

B) **How the above system can implemented using Remote Procedure Calls (RPC)?**

The software components required to implement RPC. The client accessing the service includes the stub process of each process in the service interface. The behavior of the stub process is similar to the client's local process, but instead of performing the call, it collects the process ID and parameters in the request message and sends them to the server through its communication module. When the response message arrives, it will unmarshal the result. The server process contains a scheduler, the scheduler has the server stub process and the service process of each process in the service interface. The scheduler selects one of the server stub processes based on the process identifier in the request message. Server stub process



Then define the parameters in the request message, call the corresponding service process, and collect the return value of the response message. The service process implements the process in the service interface. The client and server stub procedures and scheduler can be automatically generated by the interface compiler according to the service interface definition. RPC is usually implemented through a request-response protocol, as described in this section. The content of the request and response message

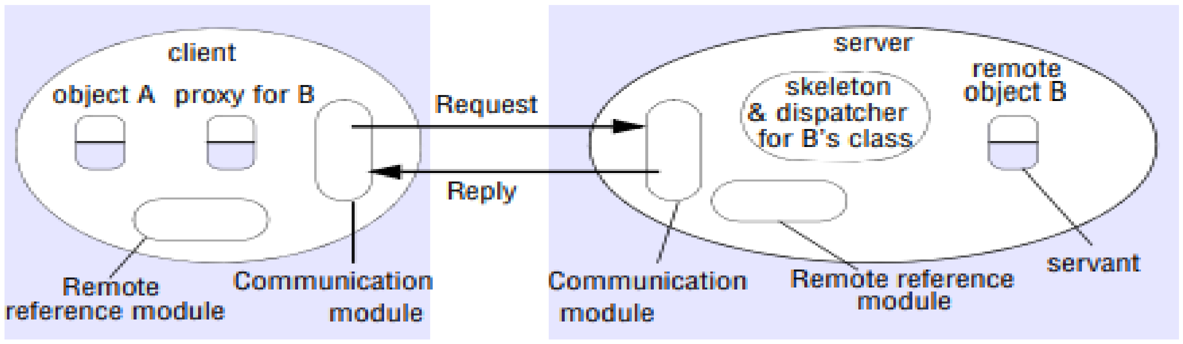
messageType	*int (0=Request, 1= Reply)*
requestId	*int*
remoteReference	*RemoteRef*
operationId	*int or Operation*
arguments	*// array of bytes*

Figure RPC can be implemented as having one of the call semantics discussed. Usually choose at least once or at most once. To this end, the communication module will implement the required design choices in terms of retransmission request, repeated processing and result retransmission, as shown in the figure.

Fault tolerance measures			*Call semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	*Maybe*
Yes	No	Re-execute procedure	*At-least-once*
Yes	Yes	Retransmit reply	*At-most-once*

<u>C)</u> How can the above system implemented using Remote Method Invocation (RMI)?

Making remote method calls involves several separate objects and modules. These are as shown

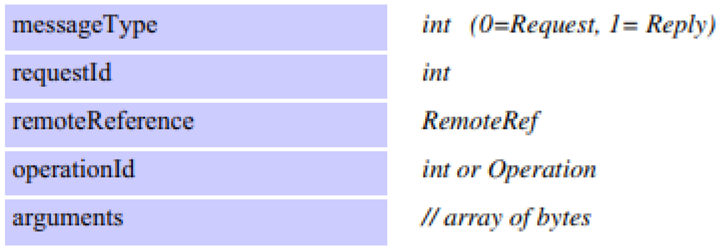


Among them, application-level object A calls methods in remote application-level object B, for which it contains a remote object reference. This section discusses the role of each component shown in the figure, first discussing the remote communication and reference modules, and then discussing the RMI software that runs them.

Then, we explore the following related topics: generating proxies, binding names to their remote object references, activating and passivation objects, and locating objects from their remote object references.

Then, we explore the following related topics: generating proxies, binding names to their remote object references, activating and passivation objects, and locating objects from their remote object references.

These two cooperating communication modules implement a request-response protocol, which transmits request and response messages between the client and the server. The content of the request and response message is shown in the figure



messageType	*int (0=Request, 1= Reply)*
requestId	*int*
remoteReference	*RemoteRef*
operationId	*int or Operation*
arguments	*// array of bytes*

The communication module uses only the first three elements, which specify the type of message, its request ID and the remote reference of the object to be called. Operation ID and all reassembly and disassembly are the focus of RMI software, as described below. The communication modules are jointly responsible for providing specified call semantics, such as at most once. The communication module of the server selects the dispatcher of the class of the object to be called by sending its local reference, which is obtained from the remote reference module to replace the remote object identifier in the request message. The next part of the RMI software will discuss the role of the scheduler.