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.**

Sessional Assignment
Course: - Distributed Computing

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

Program: - MS (CS) **Dated: 15 May 2020**

Student Name: Muhammad Faisal **Student ID#: 15139**

Class and Section: MSCS-IV Fall-18

Question: Assume you have a Client Server Environment in which the client requests 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.

a) 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)

Answer

- **Request-Reply Protocols:** As we know that the request reply protocol is consist of three steps i.e. *doOperation, getRequest and sendReply*, so whenever communication take place in client server environment, it must follow the above three steps.
  As here the environment is UDP based, so the delivery guarantees must be provided by the request-reply protocol, which may use the server reply message as an acknowledgement of the client request message.
  When a client request the server to multiply three given numbers i.e. 67, 90,34 . In the very **first** step the doOperation sends a request message to the server. The caller of doOperation is blocked until the server performs the requested operation and transmits a reply message to the client process. In the **second** step **getRequest** is used by the server process to acquire service request.
  When the server has invoked the specified operation, it then use send-reply to send the reply message to the client.
  When the reply message is received by the client, the original doOperation is unblocked and execution of the client program continuous.

- **Message Identifier:** Every message must have a unique message identifier by which it may be referenced.
  A message identifier consists of two parts:
  1. A *requestId,* which is taken from an increasing sequence of integers by the sending Process;
  2. An identifier for the sender process, for example, its port and Internet address.
  The first part makes the identifier unique to the sender, and the second part makes it Unique in the distributed system.

- **Failure model** If the three primitives *doOperation*, *getRequest* and *sendReply* are implemented over UDP datagrams, then they suffer from the same communication failures. That is:
  - They suffer from omission failures.
  - Messages are not guaranteed to be delivered in sender order.

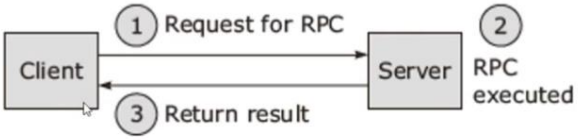
- **Timeout:** There are various options as to what *doOperation* can do after a timeout. The simplest option is to return immediately from *doOperation* with an indication to the client that the *doOperation* has failed. This is not the usual approach – the timeout may have been due to the request or reply message getting lost and in the latter case, the operation will have been performed.

- **Handle duplicate messages:** In cases when the request message is retransmitted, the server may receive it more than once. For example, the server may receive the first request message but take longer than the client's timeout to execute the command and return the reply. This can lead to the server executing an operation more than once for the same request. To avoid this, the protocol is designed to recognize successive messages (from the same client) with the same request identifier and to filter out duplicates. If the server has not yet sent the reply, it need take no special action – it will transmit the reply when it has finished executing the operation.

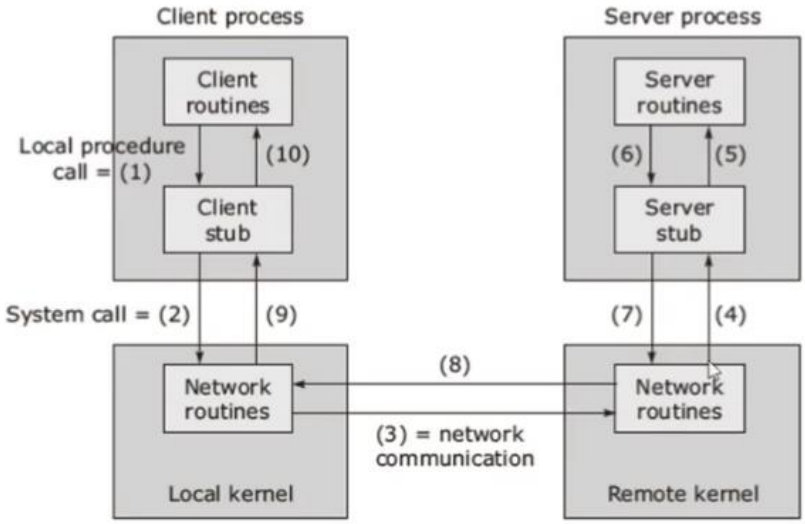
- **Lost reply messages:** if the server has already sent the reply when it receives a duplicate request it will need to execute the operation again to obtain the result, unless it has stored the result of the original execution. Some servers can execute their operations more than once and obtain the same results each time. An idempotent operation is an operation that can be performed repeatedly with the same effect as if it had been performed exactly once. A server whose operations are all idempotent need not take special measures to avoid executing its operations more than once.

b) How can the above systems implemented using Remote Procedure Calls (RPC)? (Hint: Read Section 5.3.2 in the book). (6)

Answer

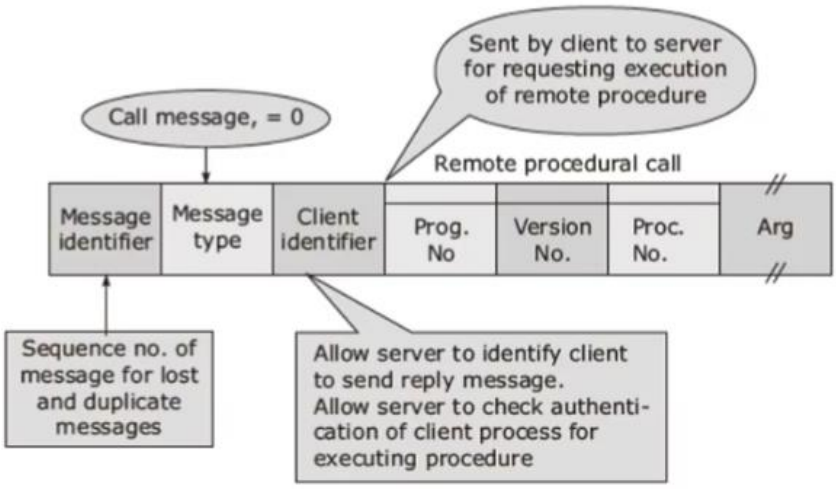


When a remote procedure call is invoked, the calling environment is suspended, the procedure parameters are transferred across the network to the environment where the procedure is to execute, and the procedure is then executed in that environment.
When the procedure finishes, the results are transferred back to the calling environment, where execution resumes as if returning from a regular procedure call.



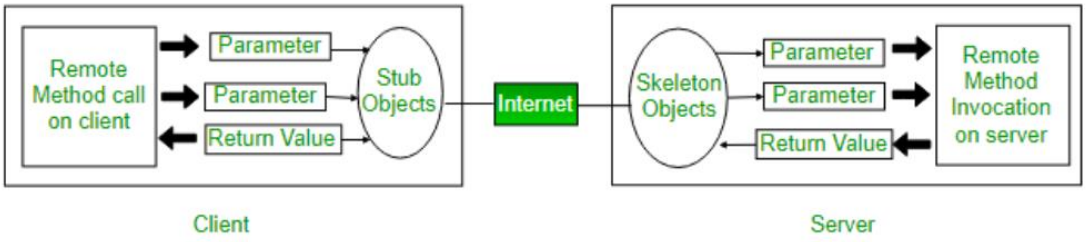
During an RPC, the following steps take place:

- The client calls the client stub. The call is a local procedure call with given three numbers for multiplication pushed onto the stack in the normal way.
- The client stub packs the procedure parameters into a message and makes a system call to send the message. The packing of the procedure parameters is called marshalling.
- The client's local OS sends the message from the client machine to the remote server machine.
- The server OS passes the incoming packets to the server stub.
- The server stub unpacks the parameters -- called unmarshalling -- from the message.
- When the server procedure is finished, it returns to the server stub, which marshals the return values into a message. The server stub then hands the message to the transport layer.
- The transport layer sends the resulting message back to the client transport layer, which hands the message back to the client stub.
- The client stub unmarshalls the return three values, and execution returns to the caller.



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

Answer: when the client makes a call to the remote object(to multiply the given three numbers), it is received by the stub which eventually passes this request to the RRL.
When the client side RRL receives the request, it invokes a method called invoke () of the object remoteRef. It passes the request to the RRL on the server side.
The RRL on the server side passes the request to the skeleton (proxy on the server) which finally invokes the required three number for multiplication on the server.
The result is passed all the way back to the client.



Working of RMI

Note: Parameter means the given three numbers i.e. 67,90,34