SAIF ULLAH ID:15598 MS(CS)

Question: 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 scenario.

a) **How the Request-Reply protocol function will be used with UDP(refer to figure 5.3 in book). How will be the message identifier 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.**

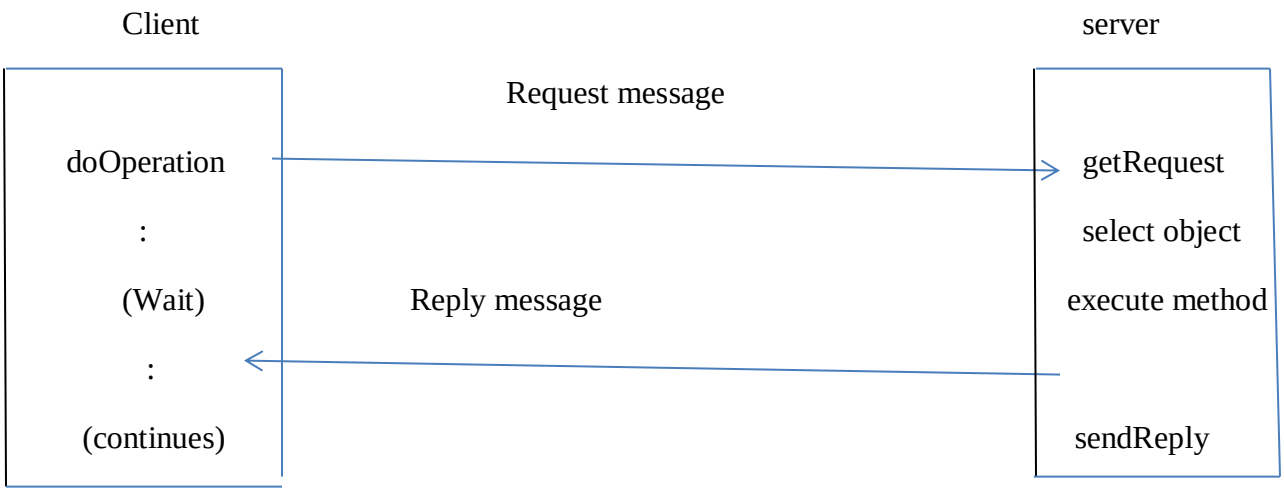
First we have to know about the request reply protocol that what is request reply protocol basically it's represent a pattern on top of the message passing and support the two way exchange of message an encountered in client-server computing. In particular such protocols provide relatively low-level support for RPC and RMI.

A protocol built over datagram avoids unnecessary overheads associated with the TCP stream protocol . in particular

- Acknowledgement are redundant, since requests are followed by replies.
- Establishing a connection involve two extra pairs of message in addition to the Pair require for a request and reply.
- Flow control is redundant of the majority of invocations, which pass only small arguments and results.

The protocol request reply protocol is based on trio of communication primitives, doOperation , getRequest and send Reply . in this case we use UDP datagram the delivery must be provided by request reply protocol, which may use the sever reply message as an acknowledgement of the client request message .

The doOperation method is used by client to invoke remote operation. First the client used the doOperation to access the remote operation we are going to used . The arguments specify the remote server and which operation to Invoke. The result contain in a byte array containing the reply.



Public byte[] doOperation (RemoteRef s, int operationaId, byte[]arguments)

Send a request message to the remote server and return the reply.

The argument specify the remote server, the operation to be invoked and the arguments of that operation.

The first arguments of the doOperation is an instance of of the class RemoteRef, which represent reference for remote server. The arguments of doOperation getting the internet address and the port no of the associated server which they want. The doOperation method send a request message to multiply the three no 67, 90, 34 to the server whose internet address and port are specified in the remote reference given in an arguments. After sending a request message doOperation invokes receive to get a replay message from which it extract the result and return it to the caller. The caller of doOperation is blocked until the server performs the requested operation and transmits a reply message to the client process.

Public byte [] getRequest();

Acquire a client request via the server port.

getRequest is used by the server process to acquire server request. When the server has invoke the specific operation like multiplying of three no 67 90 and 34 , It then uses sendReply to send the reply message to the client. When the reply message is received by the client the original doOperation is unblocked and the execution of client program is continues.

Public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);

Send the reply message reply to the client as its internet address and port.

Message Type	int (0= Request, 1=Reply)
RequestId	int
remoteReference	RemoteRef
operationId	int or Operation
Arguments	//array of bytes

The information to be transmitted in a request message or a reply message is shown above. The first field indicates whether the message is a request message or a reply message. The second field requestId contain a message identifier . third field is a remote reference. The fourth one is an identifier for the operation to be invoke and the last one is arguments.

Message Identifier:

Any scheme that involve the management of messages to provide additional properties such as reliable message delivery .

An message identifier consist of two main parts

1. Request Id, which is taking from an increasing sequence of integer by the sending process. Mean following number 67 90 and 34 has assigned a request Id according to the increasing number of sequence.
2. An identifier for the sending process e.g its port and internet address. Mean that we have the identifier for the sending process that message can be identified by its identifier.

When the value of request Id reach the maximum value for an unsigned integer (for example, $2^{32} – 1$) it rest to zero.

Failure model:

The failure model of the following message can be discussed below.

The suffer from omission failure. Mean that a mistake that consist of not doing something you should have done, or not including sometime such as an amount mean the message we have send (multiplication of three numbers) has not doing the job we actually want.

Secondly message are not guaranteed to be delivered in sender order. Mean the message we have send is not in order we want.

We can also face the crash failure mean the system is halted and we will do all the operation again.

Timeouts:

Basically a request-response or request-reply is one of the basic method computer use to communicate with each other in which the first computer sends a request for some data and the second response to the request. Usually there is a series of such interchanges until the complete message is send browsing a web page is an example of request-response communication .

There are various option as to what doOperation can do after timeout. The simplest option is to return immediately from doOperation with an indication to the client that the doOperation has failed. Mean at the specific time the message(multiplication of three numbers) cannot be delivered . The timeout have been due to the request or reply message getting lost and in the latter case the operation will have been performed .To compensate for the possibility of lost message doOperation send the request message repeatedly until either it gets a reply or its responsibly sure that the delay is due to lack of response from the server rather than lost message.

Handle Duplicate message:

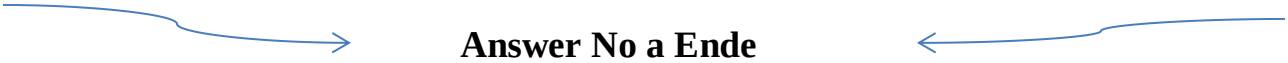
When developing message processing system there can be scenarios where an identical message can be send by an applicant more than once. In computer science if an operation is able to handle multiple instance of the same input without changing the result we say that it is idempotent.

Detecting duplicate message based on the default message id can be useful in scenario where an application sending message to a queue or topic needs to ensure that exactly one message is delivered.

To avoid this, the protocol is designed to recognize successive message (from the same client) with the same request identifier and to filter out duplicates. If the server has not yet send the reply, it need take no special action- it will transmit the reply when it has finished executing the operation.

Reply Lost Message:

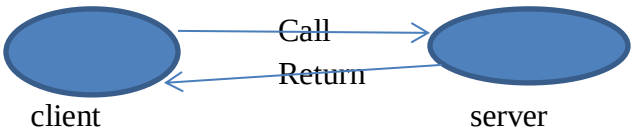
If a server has already send the reply when it receive a duplicate request it will need to execute the operation again to obtained the result, unless it has stored the result of the original execution. Some server can execute their operation more than once and obtained the same result each time. An idempotent operation is in operation that can performed repeatedly with the same effect as if it had been performed exactly once. A server whose operations is all idempotent need to take special measures to avoid executing its operations more than once.



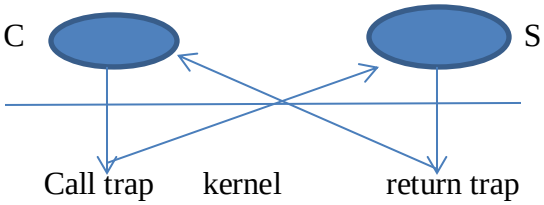
b) How can the above system implemented using Remote Procedure Calls (RPC)?

Answer b). Remote Procedure Call:

The Remote Procedure Call can be implemented for the above example are as follow.

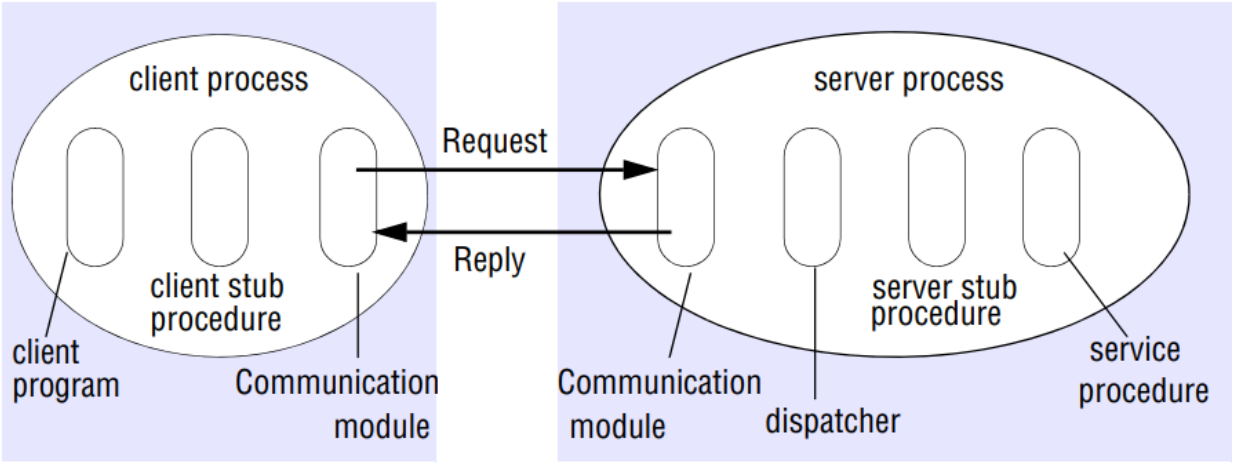


You have caller so the caller making a call server executing the call and return but under the cover let see what happen.



When the caller makes its call(multiplication of three numbers) its trap into the kernel and what the kernel does it validate the call and its copy the arguments of the call into the kernel buffers from the client address base the kernel then locates the server procedure that its need to be executed copies the arguments that it's buffer into the kernel buffer into the address base of the server in once it's done that its schedule the run the particular procedure at server point the server procedure actually start executing using the arguments of the call and performed a function that was requested by client(multiplication of 67 90 and 34). When the server procedure call is done the execution of the procedure its need to return the results (67*34*90=20502) of the procedure execution back to the client in order to do that its going to trap into the kernel (return trap) and what the kernel the does at this point is to copy the results from the address base of the server into kernel buffer and then its copied out the result from the kernel buffer into the clients address based and now at client point we have complete the result sending back to the client so the then reschedule the client who can than receive the result and go on in its marry way of executing whatever they need. More importantly all of these action is happening at runtime and that is one of the fundamental sources of performing it. So Two trap in one procedure execution that the work that has been done by run time system and order to execute this RPC.

We can also say that the software component required to implement RPC as shown in the figure below, the client that access a server (here we access the server for the operation of multiplication of three numbers) includes one stub procedure for each procedure in service interface. The stub procedure behave like a local procedure to the client, but instead of executing the call, it marshal the procedure identifier and the arguments into a request message, which send via its communication module to the server. When the reply message arrive, it unmarshals the results. The server process contain a dispatcher together with one server stub procedure and one server procedure for each procedure in the server interface. The dispatcher select one of the server stub procedures according to the procedure identifier in the request message. The server stub procedure



Then unmarshals the arguments in request message call(mean the result which client want i:e multiplication of three numbers) the corresponding service procedure and marshal the return values for the reply message. The service procedures implement the procedures in the service interface. The client in server stub procedures and the dispatcher can be guaranteed automatically by an interface compiler from the interface definition of the service.

Example:

The example given in the question can be explain in the following steps using the following cleared explained figure.

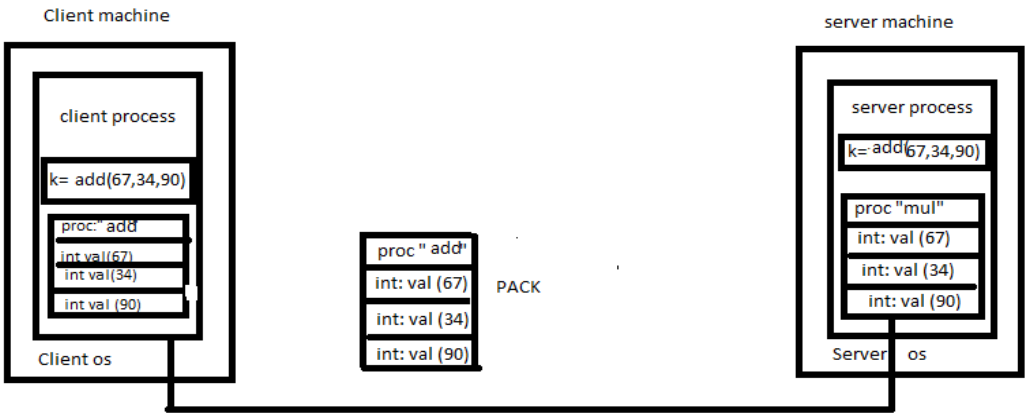


Figure b

First Step: The first step is the client call the procedure the client is calling the procedure as shown in the figure what type of procedure it is k= address of 67,34,and 90. It calls that procedure where is the procedure the procedure is on server side.so the client stub will pack and unpacks the parameters.

Second step: second step is the stub building message now it's is sending to server side

Third step: The message is sending across the network mean the two system as shown in the figure is connecting over the network.

Fourth step: The server operating system hands messages to server stub now what will the server operating system will do as shown in the figure sending message to the stub

Fifth Step: The stub unpack the message if we unpack the message we get the original message

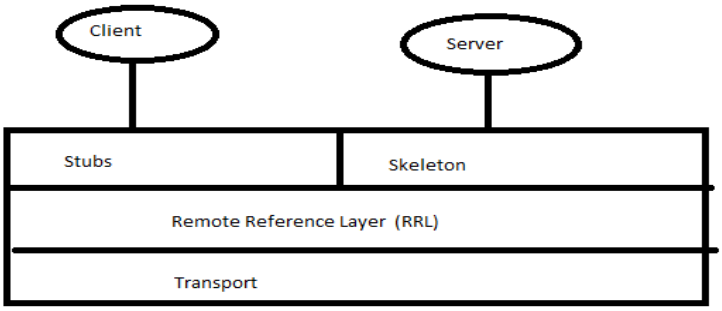
Sixth step: Then stub makes local call to address so k= address of parameters is nothing but a local call to the server process its call the procedure and send back to the client.

Answer No b Ended

C) How can the above system implemented using Remote Method Invocation (RMI)?

Answer c) Remote Method Invocation:

The core of the remote invocation call (RMI) implementation is remote reference layer (RRL). In that is a place where a lot of magic happen. The client side stub is going to initiate a remote method invocation call using remote reference layer (RRL). In all of the magic with respect to marshaling the arguments in order to send it over the network and so on is handle entirely by remote reference layer (RRL). In similarly when the result get back un marshaling the result into the data structure in which client is understand is once again done by the remote reference layer (RRL). On the server side the skeleton that exists is their for un marshaling the arguments that coming from the client. In order to un marshaling the argument the skeleton uses the remote reference layer because the remote reference layer knows how to un marshal the argument that are coming and the skeleton then make call up to the server that is implementing the remote object. Once the server is done the services the skeleton marshal the result and once again go the remote reference layer and send it over to the client and when its come back the remote reference



layer and the stub work together to deliver the result in a digestible format to the client.

The Remote reference layer is doing all the magic with respect to how the server is handle the request is it replicated is it a single server. All of these thing and many more is handle in remote reference layer (RRL). So what that mean is it allow various invocation protocol between the client and the server and all of these thing is barred in the remote reference layer (RRL).

Now what will the transport layer do the abstraction that the transport layer provide are the end point, transport, channels, and connections.

End point is nothing but a protection domain. Consist of table of remote object that it can access

Connection manager: it is that what is about all of the detail of connecting these end point together . in particular the connection management is responsible for setting up the connection, tear down the connection, listening from incoming connections and establishing the connection. When connection is established between two end points. The connection manager is also responsible for locating the dispatcher for a remote method that has been invoked on the end point .s o transport is listening in a channel when invocation come in transport layer is responsible for identifying or locating the dispatcher in the domain. Connection manager also responsible for the liveness of the connection.

RRL is the one that is declared what is the right transport to use whether it's a UDP or TCP and given that command to the connection manager.

In another word we can also say that The software components required to implement RPC are shown in Figure 5.10. The client that accesses a service includes one stub procedure for each procedure in the service interface. The stub procedure behaves like a local procedure to the client, but instead of executing the call, it marshals the procedure identifier and the arguments into a request message, which it sends via its communication module to the server. When the reply message arrives, it unmarshals the results. The server process contains a dispatcher together with one server stub procedure and one service procedure for each procedure in the service interface. The dispatcher selects one of the server stub procedures according to the procedure identifier in the request message. The server stub procedure.

Working of an RMI application:

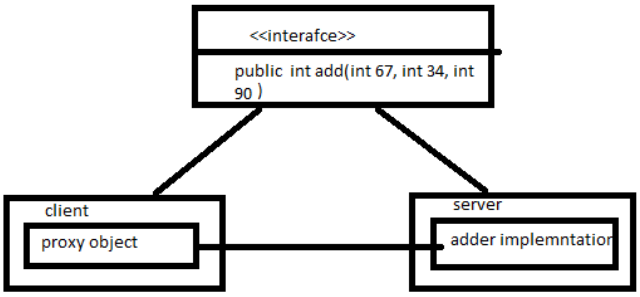
1. When the client make a call to the remote object it is received by the stub which eventually passes this request to the remote reference layer (RRL).
2. When the client side remote reference layer receive the request, it invoke a method called invoke() of the object remote Ref(). It passes the request to the Remote reference layer on the server side.
3. When the remote reference layer passes the request on the server side to the skeleton (proxy on the server)which finally invoke the required object on the sever.
4. The request is passed all the way back to the client.

Example:

Steps to write RMI program:

1. Create the remote interface
2. Provide implementation on the remote interface.
3. Compile the implementation class and create the stub and skeleton object using RMI tool.
4. Start the registry service by RMI registry tool.

5. Create and start remote application.
6. Create and start client application.



1. Create remote interface:

For creating the remote interface, extend the remote interface and declare the remote exception with all method of remote interface.

```
Import java.rmi.*;
Public interface adder extends Remote
{
        Public int add(int 67, int 34, int 90) throws remote exception
}
```

2. Provide the implementation of the remote interface:

For providing implementation we need to be either extend the unicast remote objectclass, Or use the exportobject() method

```
        Import java.rmi.*;

        Import java.rmi.server.*;

Public class adder remote extend unicast remote object implement adder

        {

          Adder Remote() throws Remote Exception

        }

        {

                Super();
```

```
}

Public int add (int 67, int 34, int 90);

Return(67 * 34 * 90);
```

3. **Create the stub and skeleton object using the Rmi tool:**
   Next step is to create stub and skeleton object using Rmi compiler. The RMI tool involve the RMI compiler and create stub and skeleton object.
   Rmic adder Remote.
4. **Start the registry service by the RMI registry tool:**
   Now start the service using RMI registry tool. If you don't specify port no: it use the default no.
   Let e.g rmi registry 7000;
5. **Create and run the server application**
6. **Create and run the client application**

An another word we can also explain the remote invocation call as follow.

Implementation of Remote Invocation call:

Several separate objects and modules are involved in achieving a remote method invocation. These are shown in Figure below, in which an application-level object A invokes a method in a remote application-level object B for which it holds a remote object reference. This section discusses the roles of each of the components shown in that figure, dealing first with the communication and remote reference modules and then with the RMI software that runs over them. We then explore the following related topics: the generation of proxies, the binding of names to their remote object references, the activation and passivation of objects and the location of objects from their remote object references.

Communication module:

The two cooperating communication modules carry out the request-reply protocol, which transmits request and reply messages between the client and server. The contents of request and reply messages are shown in Figure. The communication module uses only the first three items, which specify the message type, its requestId and the remote reference of the object to be invoked. The operationId and all the marshalling and unmarshalling are the concern of the RMI software, discussed below. The communication modules are together responsible for providing a specified invocation semantics, for example at-most-once. The communication module in the server selects the dispatcher for the class of the object to be invoked, passing on its local reference, which it gets from the remote reference module in return for the remote object

identifier in the request message. The role of dispatcher is discussed in the forthcoming section on RMI software.

Remote reference module:

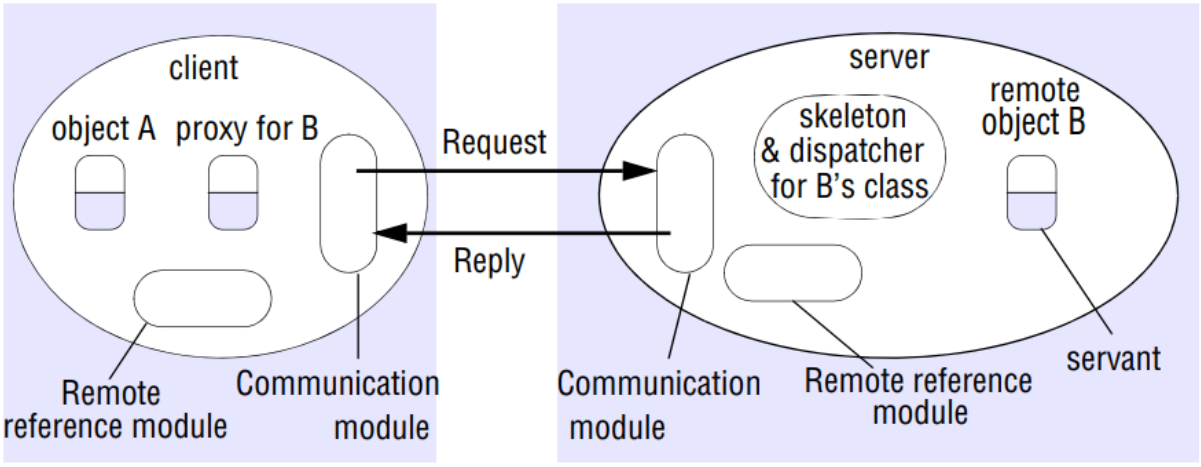
A remote reference module is responsible for translating between local and remote object references and for creating remote object references. To support its responsibilities, the remote reference module in each process has a remote object table that records the correspondence between local object references in that process and remote object references (which are system-wide). The table includes:

• An entry for all the remote objects held by the process. For example, in Figure the remote object B will be recorded in the table at the server.

• An entry for each local proxy. For example, in Figure the proxy for B will be recorded in the table at the client. The role of a proxy is discussed in the subsection on RMI software. The actions of the remote reference module are as follows:

• When a remote object is to be passed as an argument or a result for the first time, the remote reference module is asked to create a remote object reference, which it adds to its table.

• When a remote object reference arrives in a request or reply message, the remote reference module is asked for the corresponding local object reference, which may refer either to a proxy or to a remote object. In the case that the remote object reference is not in the table, the RMI software creates a new proxy and asks the remote reference module to add it to the table. This module is called by components of the RMI software when they are marshalling and un marshalling remote object references. For example, when a request message arrives, the table is used to find out which local object is to be invoked.



THE END