Remote Procedure Call.

P.K Sinah (ch-4)

- Intro to RPC
- Implement RPC
- Parameter passing.
- Call semantic and Binding.

Introduction to RPC (Remote Procedure Call).

What is Remote Procedure Call?

- It is an inter-process communication (IPC) mechanism that enable to calls a procedure that doesnot resides in the address space of the calling procedure.
- The called or remote procedure may be in the same computer as the calling process or on a different machine.

Message Passing model of Interprocess Communication :-

→ Since the caller and the called process have disjoint address spaces, the remote procedure has no access to the data and variables of the caller's environment.

→ Thus RPC uses a message-passing IPC mechanism between the caller and the called processes.

The RPC model :-

The RPC model is similar to the well-known, understood procedure call model. used for control and data transfer within a program in a following manner:-

- The caller places arguments to the procedure in some well-specified location.
- Control is then transferred to the sequence of

instructions that constitutes the body of the procedure.

- The procedure body is executed in a newly created execution environment that includes copies of arguments given in the calling instruction.

- After the procedure's execution is over, control returns to the calling point, possibly returning a result.

The RPC mechanism in an extension of this procedure call mechanism (only difference is that RPC enables a call to be made to a procedure that doesn't reside in the address space of the calling process)

The called procedure is called remote procedure. It may be on the same or different computer as the calling process.

Typical Flow of Execution in RPC :-

- Caller (Client) process sends a RPC call (request) to the called (Server) and waits (blocks) for a reply message.

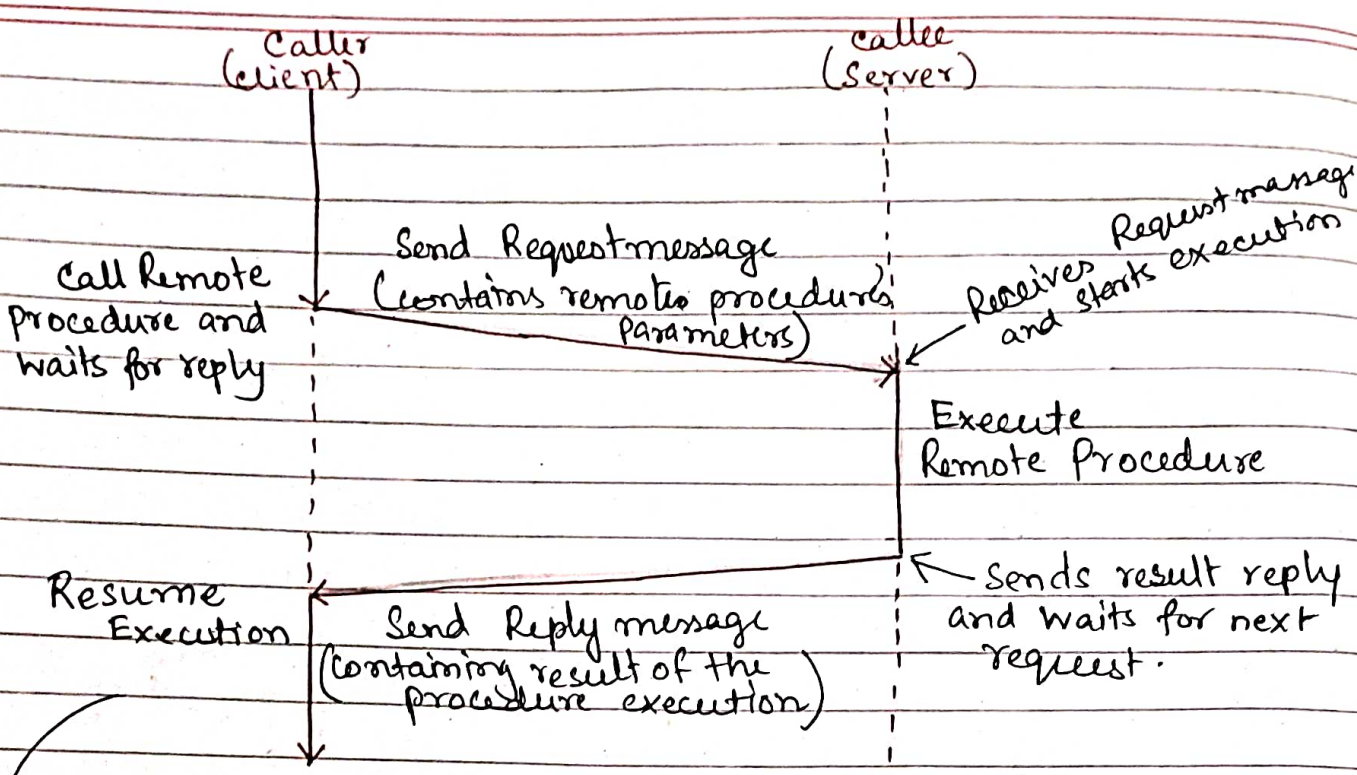
- The request message from the client contains the procedure's parameter among other things (such as caller id, etc).

- The server process executes the procedure and returns the result of execution in a reply message to the client.

- Once the reply message is sent to the client and received by the client, the result of the procedure execution is extracted.

- The execution of the client process is again resumed.

A Typical Model of RPC



Variations In Execution Model :-

- In this model of RPC, only one of the two processes is active at a given time.
- In general, RPC model makes no restrictions on the concurrency model.
- But other model of RPCs are also possible, depending on the details of parallelism of caller's and callee environment or other implementation requirement.

For eg:-

- An implementation may choose to have RPC calls to be asynchronous, so that the client may continue its execution, rather than being blocked for reply from Server.
- The Server may also create thread to process an incoming request and remain free to receive other requests.

Why RPC?

- It uses very simple call syntax.
- The semantics used for RPC calls are very similar to that of conventional call to the local procedure
- It uses well defined interface. This enables automated interface generation.
- RPC is easy to use. Simple semantics that makes it easier to build and run application for distributed system.
- It's efficient. Procedure calls are simple enough for communication to be quite rapid.
- Can be used as an IPC mechanism to communicate between different processes on the same machine.

Distribution Transparency and RPC (from Book (pg 170)).

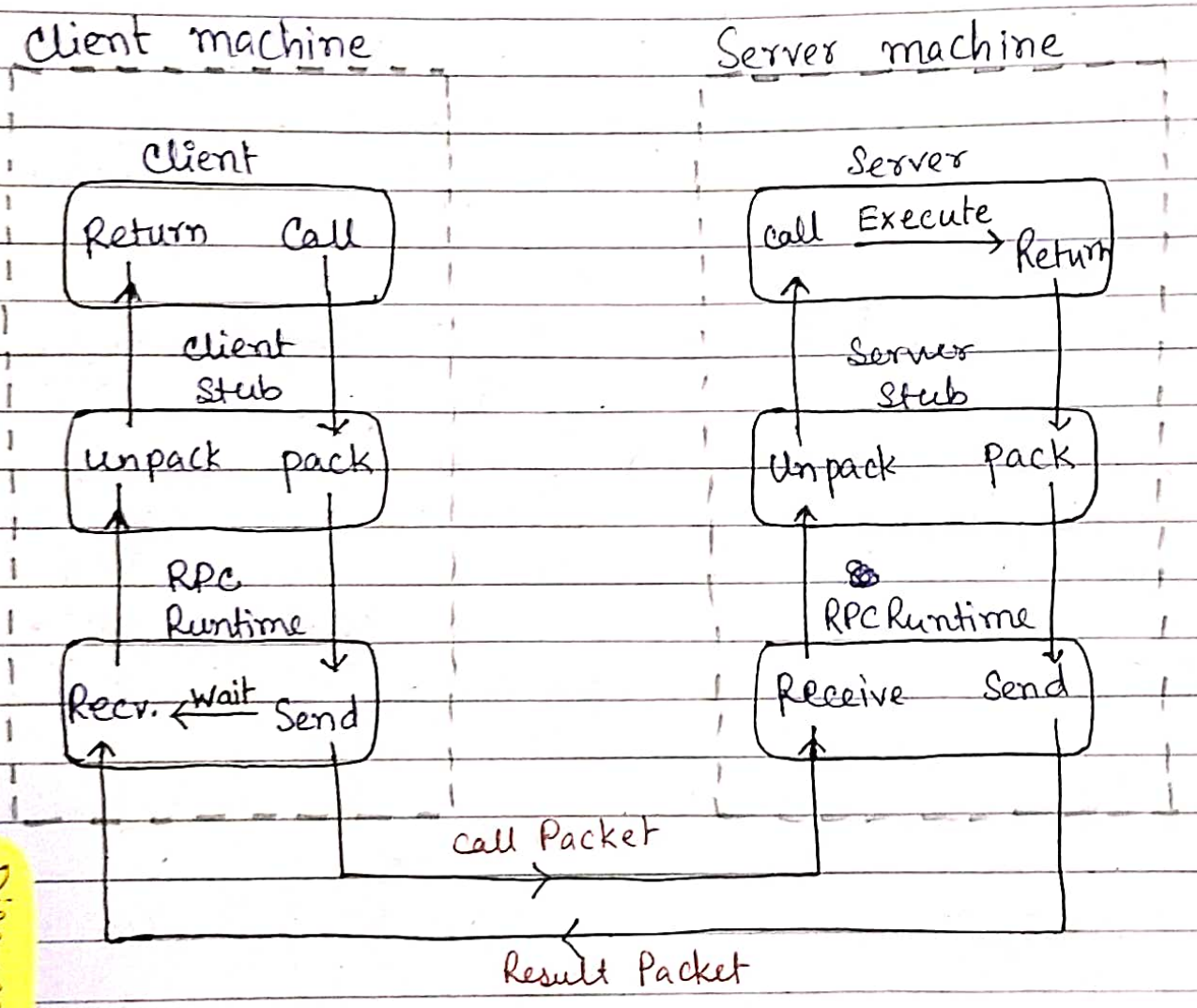
Implementing RPC :-

Implementation of RPC involves 5 elements. they are :-

- Client
- Client stub
- RPC Runtime — one in client machine, another in server machine
- Server stub
- Server.

Generic Implementation of RPC :-

- The client, client stub, and one instance of RPC Runtime executes on client machine.
- The server, server stub and another instance of RPC Runtime executes on Server machine.



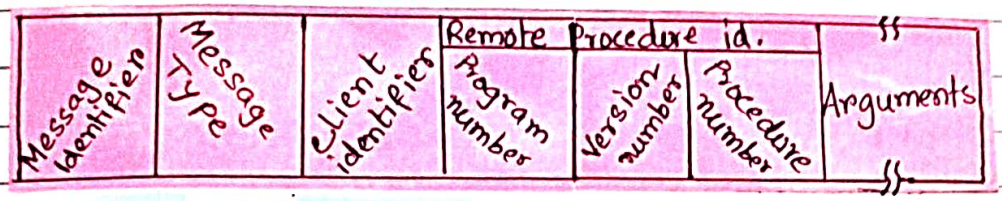
Implementation Of RPC mechanism.

1. **Client** → A user process that initiates a Remote Procedure Call. The client makes a perfectly normal local call that invokes a corresponding procedure in client stub.

Discussion on the 5 Elements of RPC

2. Client Stub → On receiving a call request from client,

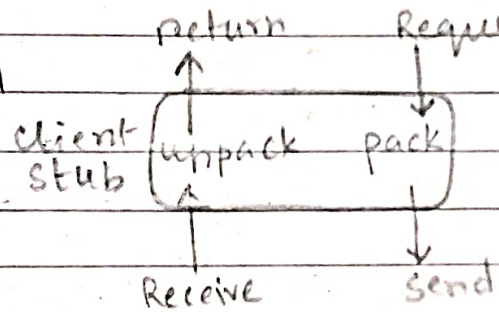
- It packs a specification of the target procedure and arguments into a message



A typical RPC call message Format

- and passes the message to the local RPC Runtime to send it to the Server Stub.

- On receiving the result of the procedure execution from the server,
  - It unpacks the result, and
  - passes it to the client.



3. RPC Runtime :-

- It handles transmission of message accross the network between client and the server.

- It is responsible for retransmissions, acknowledgement, packet routing and encryption of the message which is commonly known as Marshaling.

RPC Runtime on the Client Side :-

- The RPC Runtime on the client machine receives the call request from client stub and sends it to the server machine

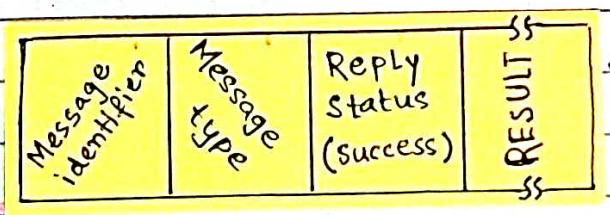
- It also receives the message containing the result of procedure execution from the server machine and passes it to the client stub.

RPC Runtime at Server Side

- It receives the call request message from the client machine and passes it to the Server stub.
- It also receives the message containing the result of the procedure execution from the server stub and sends it to the client machine.

4. Server Stub :-

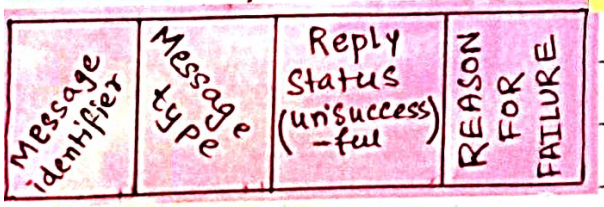
- On receiving call request message from local RPC Runtime,
  - the Server stub unpacks it., and
  - makes a perfectly normal local call to the appropiate procedure in server.
- On reciving the result of the procedure execution from the server, the server stub,
  - packs the result into a message, and



A typical RPC reply message format

↑ Successful reply message format

↓ an unsuccessful reply message format



- Passes it to the local RPC Runtime to send it to the client stub.

5. Server → On receiving a call request from server stub, the server executes the appropriate procedure and returns the result of procedure execution to the server stub.

Summary of Steps in RPC

- The client procedure calls the client stub.
- The client stub builds a message and passes to the RPC runtime at client.
- The client's RPC Runtime sends the message to the remote site.
- RPC Runtime in the Remote site passes message to the server stub.
- The server stub unpacks the parameters and passes it to the server.
- The server executes the procedure and returns the result to the server stub.
- The server stub packs it in a message and passes it to its RPC Runtime.
- RPC Runtime at server's site sends the message to the client's node.
- Client's RPC runtime receives it and passes the message to the client stub.
- The client stub unpacks the result and sends it to the client.

Transparency in RPC :-

- The beauty of the whole scheme is that the client is ignorant of the fact that the work was done remotely and not by the the local Kernel.

- When the client gets control following the

procedure call that it made, all it knows is that, the result of the procedure execution are available to it.

- Thus for client, remote services are accessed by making ordinary (Local) procedure call, and not by using send and receive primitives.

- All the details of the message passing are guarded by the client and server stubs, making the steps involved in message passing, invisible to both client and server.

Stub Generation

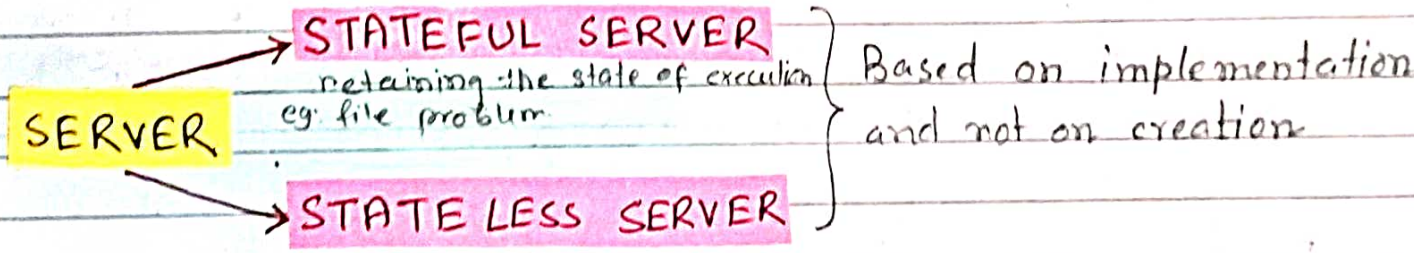
Manual	Automatic
The RPC implementor provides a set of translation functions from which users can construct his or her own stubs. The method is simple to implement and can handle very complex parameter types	This is the most commonly used method for stub generation. It uses Interface definition Language (IDL) that is used to define the interface between a client and a server.

Marshalling Arguments and Results:-

- Remote procedure calls involve the transfer of arguments from client to server process and transfer of result from server to client process.

- Transfer of these data between the two computers often requires encoding and decoding of the message data.
- This operation is know as marshaling and un-marshaling in case of RPCs.
- This data encoding process involves the conversion of program objects into a stream form which is suitable for transmission and placing them into a message buffer.
- The decoding process involves the reconstruction of program objects from the message data that was received in stream form.

Server implementation



- **Stateful Server** :- (Server Retaining the state of execution).

→ A stateful server retains clients' state information from one RPC to the next.

→ In case of two or more subsequent calls by a client to the stateful server, some state information (till which state the process execution was done) from the service performed for the client in the first call, is stored by the server process.

→ This same information is utilized for executing the second call.

- **Stateless Server** :- (does not maintain client state info)

→ A stateless server does not maintain any client state information.

→ Every request from the client must be accompanied with all the necessary parameters to successfully carry out the desired operation.

✱ Stateful or Stateless ?

- Easier to code for a stateful server.
- It relieves the client from saving the state information.

→ Stateless servers have a distinct advantage in event of failure.

→ Because if a stateful server crashes, the state information may be lost and the client process might continue unaware of the crash, producing inconsistent result.

✱ Passing the Parameters :-

- Call by Value
- Call by Reference
- Call by Object Reference
- Call by Move
- Call by Visit

(marked in book page-183).

Call Semantics and Binding

Call Semantics :-

→ What the client can assume about execution of remote procedure call.

- We know that in RPC the caller and the callee processes are possibly located on different nodes.
- Thus it is possible for either of the node to fail independently and later to be restarted.
- This can happens due to one or more of the following Reasons :-

- The call message gets lost.
- The response message gets lost.
- The callee node crashes and is restarted.
- The caller nodes crashes and is restarted.

Why call semantics is #important ?

* Element of a caller's node that is involved in the RPC must contain necessary code for handling failures.

- the client code itself should not be forced to deal with these failures, thus faliure-handling code is a part of the RPC runtime.

- Call semantic that decides if RPC is to be executed for faults depends on the RPC Runtime code.

- This provids flexibity to the application programmers to select from different possible call semantics supported by the system.