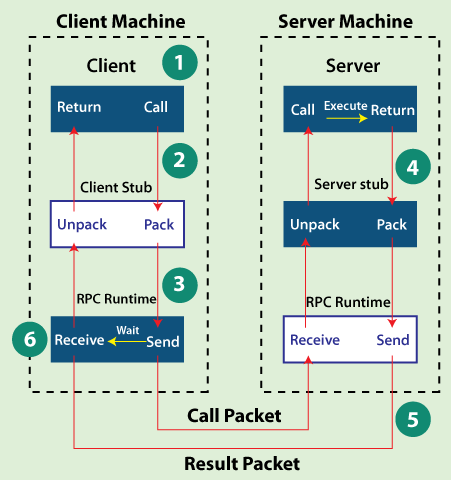
**Aim:** Implementing Client Server using Remote Procedure Call

**Theory:**

Remote Procedure Call or RPC is a powerful technique for constructing distributed, client-server-based applications. It is also known as a function call or a subroutine call. A remote procedure call is when a computer program causes a procedure to execute in a different address space, coded as a local procedure call, without the programmer explicitly stating the details for the remote interaction. The programmer writes essentially the same code whether the subroutine is local to the executing program or remote. This is a form of client-server interaction implemented via a request-response message-passing system.

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.



A remote procedure call (RPC) works in the following steps in an operating system:

**Step 1:** The client, client stub, and RPC run time execute on the client machine.

**Step 2:** A client starts a client stub process by passing parameters in the usual way. The packing of the procedure parameters is called ***marshalling***. The client stub stores within the client's own address space, and it also asks the local RPC Runtime to send back to the server stub.

**Step 3:** In this stage, the user can access RPC by making regular Local Procedural Call. RPC Runtime manages the transmission of messages between the network across client and server, and it also performs the job of retransmission, acknowledgment, routing, and encryption.

**Step 4:** After completing the server procedure, it returns to the server stub, which packs (marshalls) the return values into a message. The server stub then sends a message back to the transport layer.

**Step 5:** In this step, the transport layer sends back the result message to the client transport layer, which returns back a message to the client stub.

**Step 6:** In this stage, the client stub demarshalls (unpack) the return parameters in the resulting packet, and the execution process returns to the caller.