Remote Procedure Call

Basics and Principle of Operations

1

Outline

- Introduction to RPC
- Implementing RPC
- Passing the Parameters
- Call Semantics in RPC
- Client Server Binding

16 July 2024

2

Outline

- Introduction to RPC
- Implementing RPC
- Passing the Parameters
- Call Semantics in RPC
- Client Server Binding

16 July 2024

3

What is Remote Procedure Call?

- RPC is an inter-process communication (IPC) mechanism that enables to call a procedure that does not reside in the address space of the calling process.
- The called or remote procedure may be in the same computer as the calling process or on a different machine.

16 July 2024

4

Message passing IPC

- Since the caller and the called processes have disjoint address spaces, the remote procedure has no access to data and variables of the caller's environment.
- RPC uses a message-passing IPC mechanism between the caller and called processes.

16 July 2024

5

Typical Flow of Execution in RPC

- Caller (client) process sends a RPC call request to the called (server) and waits (blocks) for a reply.
- The request message from the client contains the remote procedure's parameters, caller id and other relevant information.

16 July 2024

6

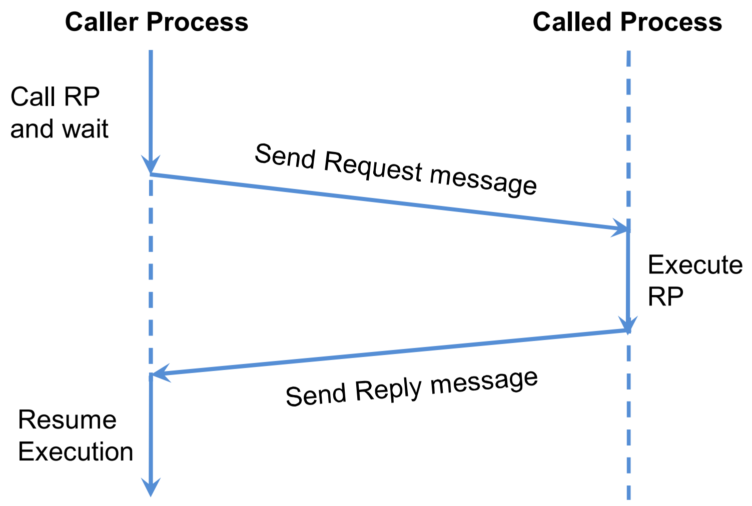
Typical Flow of Execution in RPC

- The server process executes the remote procedure called and returns the result in a reply message.
- Once the reply message is received at client, the result of procedure execution is extracted.
- Execution of the client process resumes.

16 July 2024

7

Typical Flow of Execution in RPC



8

Variations in Execution Model

- In this model, only one of the two processes is active at any given instance.
- In general, RPC protocol makes no such restrictions on the concurrency model.
- Other execution models are possible based on the RPC implementation requirements.

16 July 2024

9

Variations in Execution Model

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

16 July 2024

10

Why RPC?

- The wide usage of RPC is due to the following features:
  - RPC uses very simple call syntax.
  - The semantics used for RPC calls are quite similar to conventional call to local procedures.
  - RPC uses a well-defined interface. This enables automated interface generation.

Why RPC?

- It's efficient. Procedure calls are simple enough for communication to be quite rapid.
- It can be used as an IPC mechanism to communicate between processes on different machines as well as between different processes on the same machine.

Distribution Transparency and RPC

- A transparent RPC mechanism is one in which local and remote procedures are effectively indistinguishable to programmers.
- It is easier to achieve syntactic transparency for RPC as the basic functioning of RPC closely follow that of local procedure calls.

16 July 2024

13

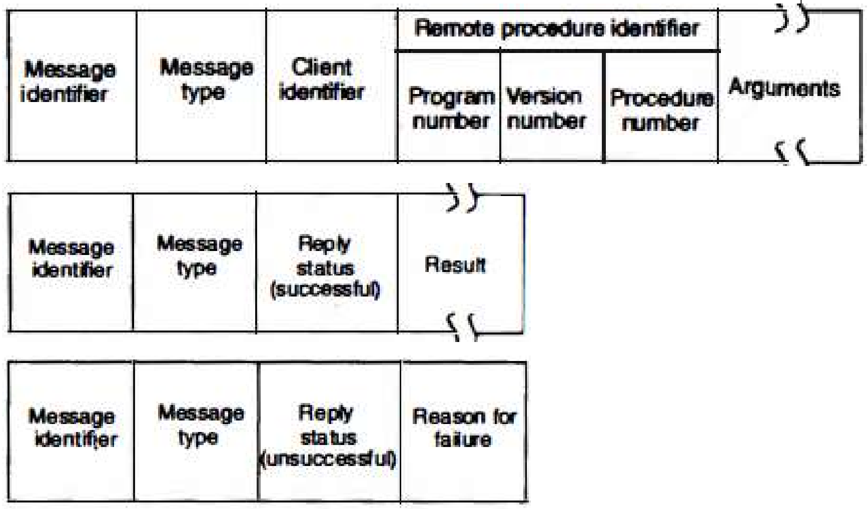
Outline

- Introduction to RPC
- Implementing RPC
- Passing the Parameters
- Call Semantics in RPC
- Client Server Binding

16 July 2024

14

Message Formats in RPC



15

Implementing RPC

- Implementation of RPC involve 5 elements. These are
  - Client
  - Client stub
  - RPC Runtime
  - Server stub
  - Server

16 July 2024

16

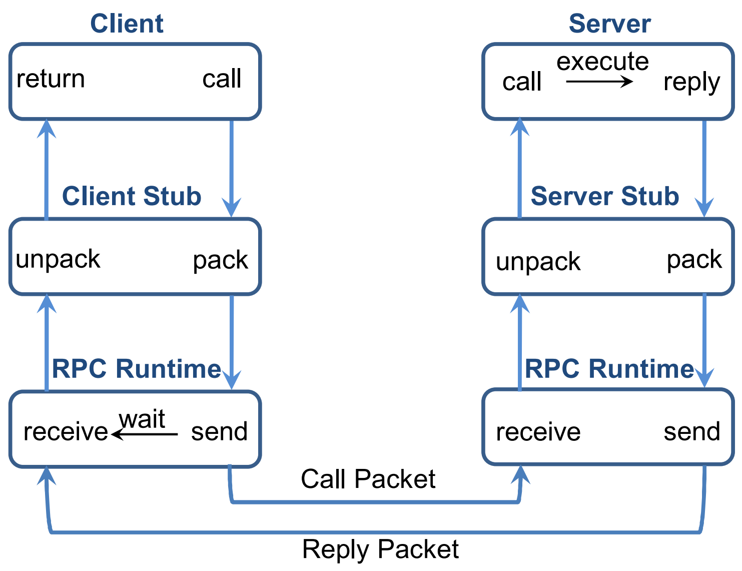
Implementing RPC

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

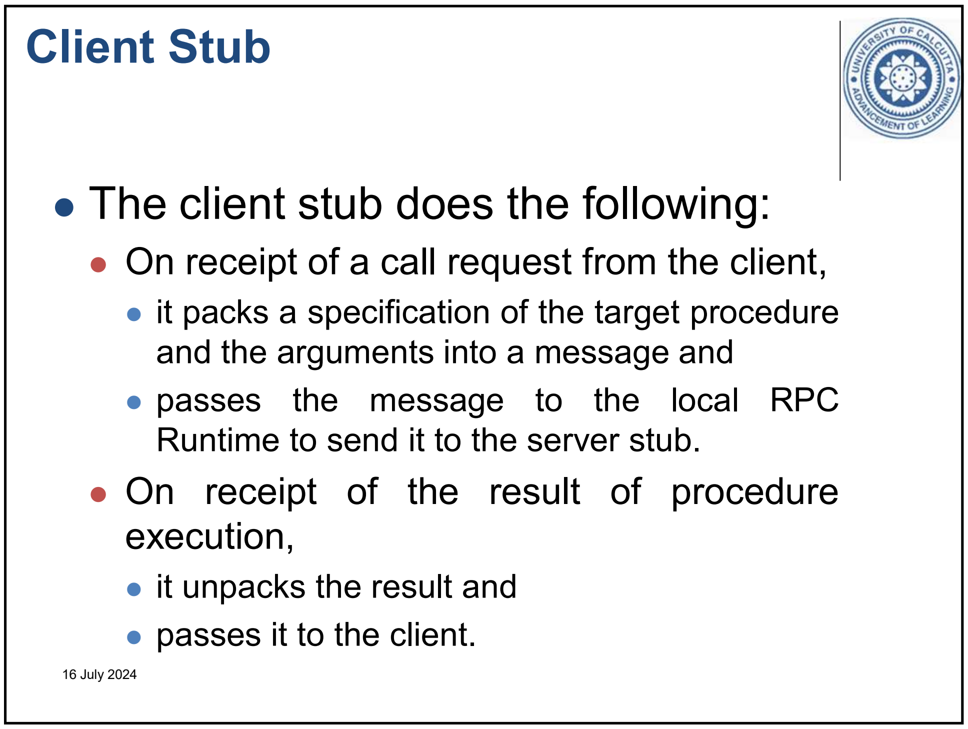
16 July 2024

17

Generic Implementation of RPC



18

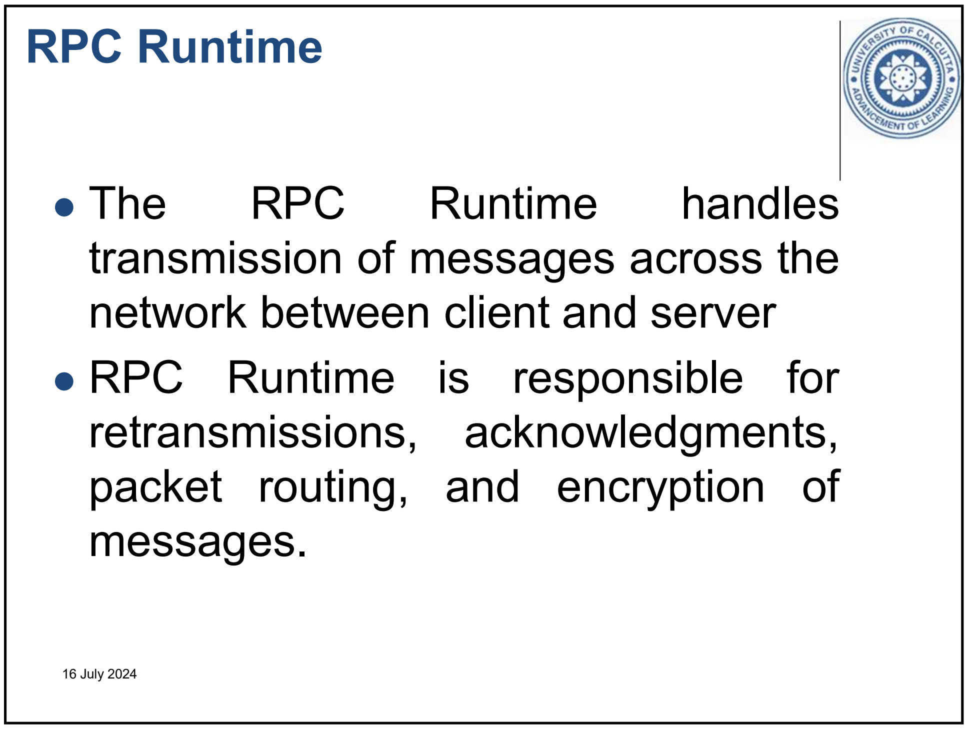


Client Stub

- The client stub does the following:
  - On receipt of a call request from the client,
    - it packs a specification of the target procedure and the arguments into a message and
    - passes the message to the local RPC Runtime to send it to the server stub.
  - On receipt of the result of procedure execution,
    - it unpacks the result and
    - passes it to the client.

16 July 2024

19



RPC Runtime

- The RPC Runtime handles transmission of messages across the network between client and server
- RPC Runtime is responsible for retransmissions, acknowledgments, packet routing, and encryption of messages.

16 July 2024

20

RPC Runtime at Client Side

- The RPC Runtime on the client machine receives the call request message from the 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.

16 July 2024

21

RPC Runtime at Server Side

- On the other hand, the RPC Runtime on the server machine receives the message containing the result of procedure execution from the server stub and sends it to the client.
- It also receives the call request message from the client machine and passes it to the server stub.

16 July 2024

22

Server Stub

- The server stub does the following:
  - On receipt of the call request message from the local RPC Runtime,
    - the server stub unpacks it and
    - makes a local call to the appropriate procedure.
  - On receipt of the result of procedure execution from the server, the server stub
    - packs the result into a message and
    - Passes it to the local RPC Runtime to send it to the client stub.

16 July 2024

23

Summary of Steps in RPC

- The client procedure calls the client stub.
- The client stub builds a message and passes to 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.

16 July 2024

24

Summary of Steps in RPC

- 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.
- Server stub packs it in a message and passes it its RPC Runtime.

16 July 2024

25

Summary of Steps in RPC

- RPC Runtime at the server's site sends the message to the client node.
- Client's RPC Runtime passes the message to the client stub.
- The client stub unpacks the result and returns to the client.

16 July 2024

26

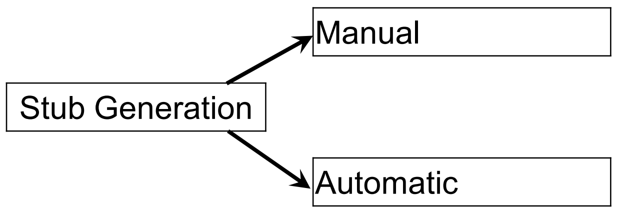
Transparency in RPC

- The beauty of the whole scheme is the ignorance on the part of the client that the work was done remotely instead of by the local kernel.
- When the client gets control following the procedure call that it made, all it knows is that the results of the procedure execution are available.

Transparency in RPC

- Thus, for the client, remote services are accessed by making ordinary (local) procedure calls, not by using the 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



16 July 2024

29

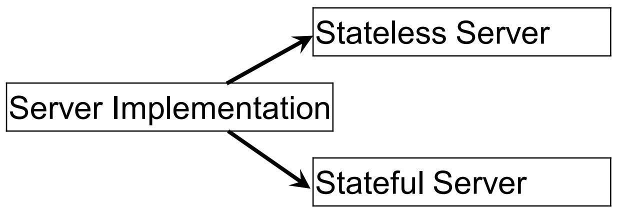
Marshalling Arguments and Results

- Transfer of data between two computers often requires encoding and decoding of the message data.
- This operation is known as marshaling and un-marshaling in case of RPCs.

16 July 2024

30

Server Implementation



16 July 2024

31

Stateful Server

- A stateful server retains clients' state information from one RPC to the next.
- In case of two subsequent calls by a client to a stateful server, some state information from the service performed for the client in the first call is stored by the server process.
- The same is utilized for executing the second call.

32

Stateless Server

- A stateless server does not maintain any client state information.
- Every request from a client must be accompanied with all the necessary parameters to successfully carry out the desired operation.

Stateless or Stateful

- It’s easier to code for a stateful server
- A statelful server relieves the clients from packing the state information again and again
- Stateless servers have a distinct advantage in the event of a failure.
- If a stateful server crashes, the state information may be lost and the client process might continue unaware of the crash, producing inconsistent results.

Outline

- Introduction to RPC
- Implementing RPC
- Passing the Parameters
- Call Semantics in RPC
- Client Server Binding

16 July 2024

35

Parameter Passing

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

16 July 2024

36

Call by Value

- In call-by-value, the parameters are copied into a message and sent to server.
- Good for simple data, e.g., integers, counters, small arrays, and so on.
- Passing larger data types such as trees. graphs, large object codes, etc. will require huge time.
- Therefore, this is not suitable for passing parameters involving voluminous data.

Call by Reference

- Here, the client and the server exist in different address spaces. Hence, passing pointers or passing parameters by reference is meaningless.
- This may be feasible for distributed systems having distributed shared-memory mechanisms.

Call by Object Reference

- In an object-based system that uses the RPC for object invocation, call-by-reference is implemented as call-by-object-reference.
- This is because in an object-based system, the value of a variable is a reference to an object.
- Thus, it is this reference (the object name) that is passed in for invocation.

16 July 2024

39

Call by Move

- In call-by-move, a parameter is passed by reference, as in the method of call-by-object-reference.
- However, at the time of the call, the parameter object is moved to the destination node i.e., at the server site.
- After the call, the argument object may either return to the caller's node or remain at the remote node.

16 July 2024

40

Call by Visit or Call by Move?

- These two options are actually known as call-by-visit and call-by-move, respectively.

16 July 2024

41

Outline

- Introduction to RPC
- Implementing RPC
- Passing the Parameters
- Call Semantics in RPC
- Client Server Binding

16 July 2024

42

Why Call Semantics is important?

- Normal functioning of RPC may get disrupted due to:
  - The call message gets lost.
  - The response message gets lost.
  - The client node crashes and restarts.
  - The server node crashes and restarts.

Why Call Semantics is important?

- Some element of client node that must handle these exceptions
- Obviously, the client code itself should not be forced to deal with these failures.
- Therefore, the failure-handling code is generally a part of RPC Runtime.

Why Call Semantics is important?

- Call semantics of RPC system that decides if the RP is to be executed for faults depends on the RPC Runtime code.
- This would provide some flexibility to the application programmers to select from different possible call semantics supported by an RPC system.

16 July 2024

45

Different Call Semantics

- May be call semantics
- Last one call semantics
- Last of many call semantics
- At-least-once call semantics
- Exactly once call semantics

16 July 2024

46

May be Call Semantics

- This is the weakest call semantics and is not really appropriate to RPC - mentioned here for completeness.
- In order to prevent the caller from waiting indefinitely for a response from the remote process, a timeout mechanism is used.

16 July 2024

47

May be Call Semantics

- The caller waits until a preset timeout period and then continues execution
- Hence, this does not guarantee anything about receipt of reply.
- This may be adequate for some applications in which the response message is not important for the caller.

16 July 2024

48

Last One Call Semantics

- This call semantics uses the idea of retransmitting the call message based on timeouts until a response is received by the caller.
- The calling of the remote procedure, its execution in the server space, and the reply to the caller will be repeated until the result of procedure execution is received by the caller.

16 July 2024

49

Last One Call Semantics

- Clearly, the effect of the last execution should persist.
- The earlier (abandoned) calls may have had side effects that survived the crash.
- Hence, this semantics is called last-one call semantics.

16 July 2024

50

Orphan Call

- An orphan call is one whose parent (caller) has expired due to a node crash.

16 July 2024

51

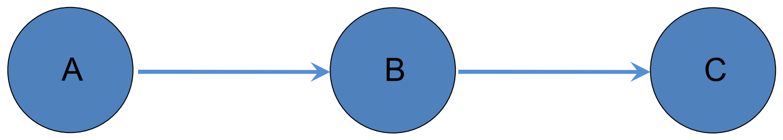
Orphan Call

- The orphan calls must be terminated before restarting the crashed processes to achieve last-one semantics.
- This is normally done either by waiting for them to finish or by tracking them down and killing them.
- This is not an easy job.

16 July 2024

52

Orphan Call



- By definition, none of the two calls to procedure C are orphan calls as site for B has never crashed

16 July 2024

53

Achieving Last One Call Semantics

- Achieving last-one semantics in presence of crashes turns out to be tricky for nested RPCs that involve more than two procedures in as many nodes.
- The basic difficulty in achieving last-one semantics in such cases is caused by orphan calls.

16 July 2024

54

Last of Many Call Semantics

- This is similar to the last-one semantics except that the orphan calls are identified and dropped.
- A simple way to drop orphan calls is to use call identifiers to uniquely identify each call.
- When a call is repeated, it is assigned a new call identifier.

16 July 2024

55

Last of Many Call Semantics

- Each response message has the corresponding call identifier associated with it.
- A caller accepts a response only if the call identifier matches with that of the most recently repeated call.
- Otherwise, it ignores the response message.

16 July 2024

56

At Least Once Call Semantics

- This is even weaker than the last-of-many call semantics.
- It guarantees that the call executes one or more times but does not care which result is returned to the caller.
  - i.e., for nested calls, if there are any orphan calls, it takes the result of the first response message, irrespective of whether or not the accepted response is from an orphan.

At Most Once Call Semantics

- This is the strongest and the most desirable call semantics
- This eliminates the possibility of a procedure being executed more than once no matter how many times the call is retransmitted.

At Most Once Call Semantics

- Call semantics like last-one, last-of-many or at-least-once forces the application programmer to design idempotent interfaces
  - Interface that guarantee that if even a procedure is executed more than once with the same parameters, the same results and side effects will be produced.

16 July 2024

59

At Most Once Call Semantics

- The programmer is relieved of the burden of implementing the server procedure in an idempotent manner for exactly-once semantics.
- The call semantics itself takes care of executing the procedure only once.

16 July 2024

60

At Most Once Call Semantics

- The implementation of exactly-once call semantics could be based on the following:
  - Use of timeouts
  - Re-transmissions of call and reply messages
  - Call identifiers with the same identifier for repeated calls
  - Reply cache in the server.

16 July 2024

61

Communication Protocols for RPC

- Request (R)
- Request Reply (RR)
- Request Reply Acknowledgement (RRA)

16 July 2024

62

Request Protocol (R) for RPC

- It is used in RPCs when
  - RP has nothing to return as the result and,
  - the client requires no confirmation that the procedure has been executed.
- The client proceeds immediately after sending the request message as there is no need to wait for a reply message.
- Supports may-be call semantics.

Request Protocol (R) for RPC

- An RPC that uses the R protocol is called asynchronous RPC.
- Asynchronous RPC helps in improving the combined performance of both the client and the server in those distributed applications in which the client does not need a reply to each request.

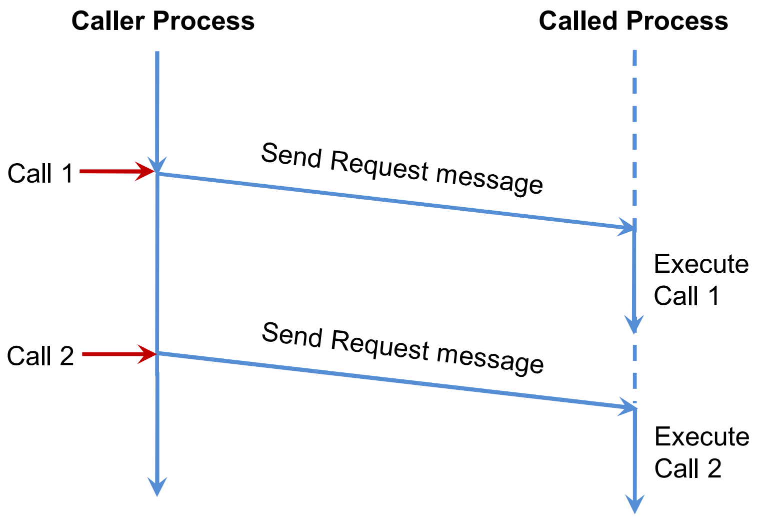
Request Protocol (R) for RPC

- Client performance is improved because the client is not blocked and can immediately continue to do other work after making the call.
- On the other hand, server performance is improved because the server need not generate and send any reply for the request.

16 July 2024

65

Flow of Execution in R Protocol



66

Request Protocol (R) for RPC

- Asynchronous RPC does not require retransmission of request messages
- However, if a reliable, connection-oriented transport protocol e.g., TCP is used then issue of retransmitting the request message does not occur at all because the reliable transport protocol delivers it in any case.

Request Protocol (R) for RPC

- Applications using asynchronous RPC with unreliable transport protocol, e.g., UDP, must be prepared to handle this situation.
- Asynchronous RPCs with unreliable transport protocol may be useful for implementing periodic update services, like status updating.

Request Reply Protocol (RR) for RPC

- The protocol is based on the idea of using implicit acknowledgment to eliminate explicit acknowledgment messages. In this protocol:
  - A server's reply message is regarded as an acknowledgment to client's request message.
  - Subsequent call packet from a client is regarded as an acknowledgment of the server's reply message for the previous call from the same client.

16 July 2024

69

Request Reply Protocol (RR) for RPC

- RR protocol requires transmission of only two packets per call
- It is useful for simple RPCs.
- A simple RPC is one in which all the arguments as well as all the results fit in a single packet buffer and the duration of a call and the interval between calls are both short.

16 July 2024

70

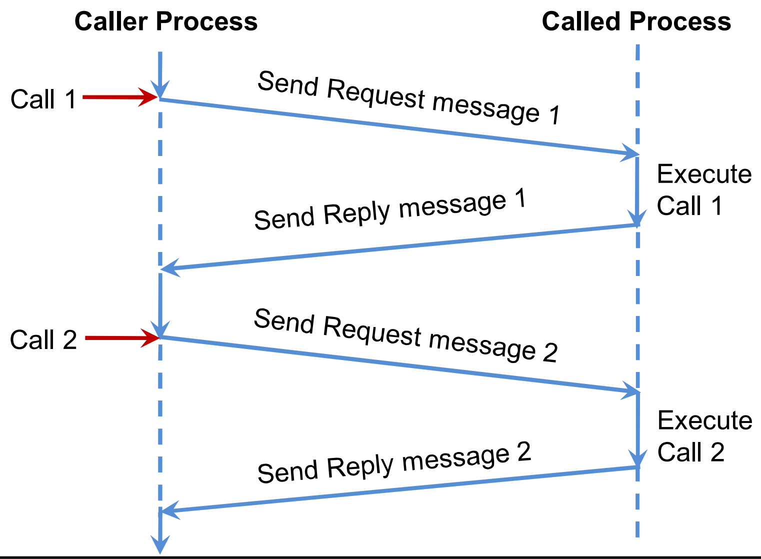
Request Reply Protocol (RR) for RPC

- RR protocol requires transmission of only two packets per call
- It is useful for simple RPCs.
- A simple RPC is one in which all the arguments as well as all the results fit in a single packet buffer and the duration of a call and the interval between calls are both short.

16 July 2024

71

Flow of Execution in RR Protocol



72

RR Protocol and Call Semantics

- A client retransmits its call request if it does not get the reply message within a predetermined timeout period.
- Thus, if duplicate request messages are not filtered out, then RR protocol, combined with this failure handling technique, supports the at-least-once call semantics.

16 July 2024

73

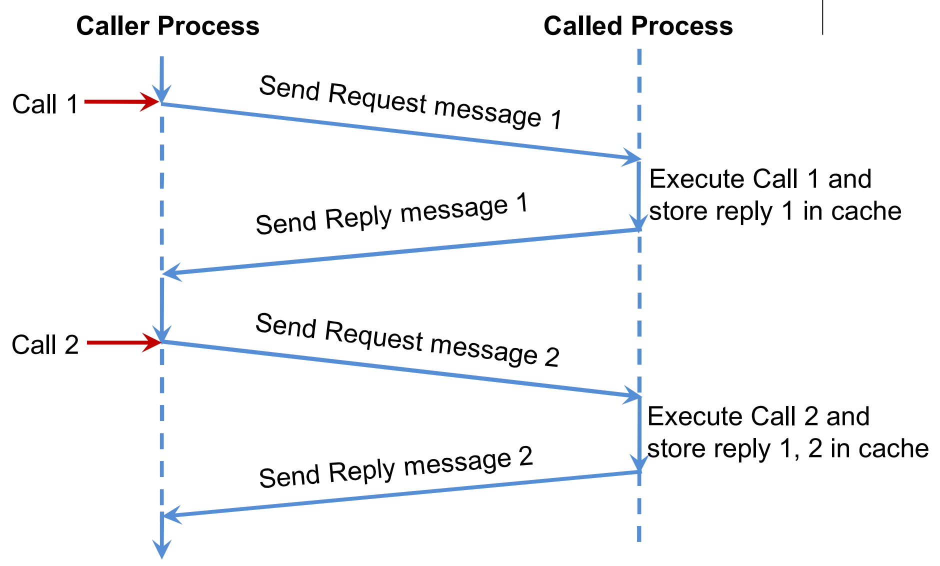
RR Protocol and Call Semantics

- However, servers can support exactly-once call semantics by storing records of the replies in a reply cache
- Using the cache, duplicate request messages can be filtered out and retransmission of the corresponding reply messages saves from multiple execution for duplicate requests.

16 July 2024

74

Flow of Execution in RR Protocol



75

Request Reply Acknowledgement Protocol (RRA) for RPC

- The implementation of exactly-once call semantics with RR protocol requires the server to store the replies in a reply cache.
- If a server has a large number of clients, then large amount of data needs to be stored in cache.

16 July 2024

76

Request Reply Acknowledgement Protocol (RRA) for RPC

- In some implementations, servers restrict the storage by discarding replies after a limited period of time.
- However, this approach is not fully reliable as sometimes it may lead to the loss of those replies that have not yet been successfully delivered to their clients.

16 July 2024

77

Request Reply Acknowledgement Protocol (RRA) for RPC

- In RRA protocol clients acknowledge receipt of reply messages.
- The server deletes an information from its reply cache only after receiving an acknowledgment for it from the client.
- RRA protocol involves transmission of three messages per call.

16 July 2024

78

Request Reply Acknowledgement Protocol (RRA) for RPC

- In RRA protocol, there is a possibility that the acknowledgment message may itself get lost.
- Implementation of the RRA protocol requires that a unique message identifiers is associated with request messages.

16 July 2024

79

Request Reply Acknowledgement Protocol (RRA) for RPC

- Each reply contains the message identifier of corresponding request message and each acknowledgment also contains the same message identifier.
- This helps in matching a reply with its corresponding request and an acknowledgment to its corresponding reply.

80

Request Reply Acknowledgement Protocol (RRA) for RPC

- A client acknowledges a reply only if it has received replies to all the requests previous to the request corresponding to this reply.
- Thus an acknowledgment message is interpreted as acknowledging the receipt of all reply messages till then.
- Thus loss of an acknowledgment message is harmless for RRA.

81

Outline

- Introduction to RPC
- Implementing RPC
- Passing the Parameters
- Call Semantics in RPC
- Client Server Binding

16 July 2024

82

Client Server Binding

- The process by which a client connects with a server so that calls can take place is known as binding.
- It is necessary for a client to know the location of a server before a RPC can take place between them.

Client Server Binding

- Typically, the servers register through a binding mechanism to announce their willingness to “export” the services.
- Clients, on the other hand, "import" operations, asking the RPC Runtime system to locate a server using binding mechanism.

Client Server Binding Issues

- How does a client specify a server to which it wants to get bound?
- How does the binding process locate the specified server?
- When is it proper to bind a client to a server?

16 July 2024

85

Client Server Binding Issues

- Is it possible for a client to change binding during execution?
- Can a client be simultaneously bound to multiple servers that provide the same service?

16 July 2024

86

Locating the Server

- Broadcasting
  - Client broadcasts service request message to all the nodes.
  - The nodes on which the desired server is located return a response message.
- Using Binding Agent
  - A binding agent is basically a name server used to bind a client to a server.

16 July 2024

87

Locating a Server by Broadcasting

- Here, the desired server may be replicated on several nodes.
- In such case the client will receive responses from all these nodes.
- Normally, the first response received at the client's node is selected.

16 July 2024

88

Locating a Server by Broadcasting

- All other responses are discarded.
- This method is easy to implement and is suitable for use for small networks.
- However, the method is expensive for large networks due to high message overhead.

Binding Agent

- A binding agent maintains a binding table, which is a mapping of a server's interface name to its locations.
- All servers register themselves with the binding agent as part of their initialization process.
- A server gives the binder its identification information and a handle to locate it as part of registration.

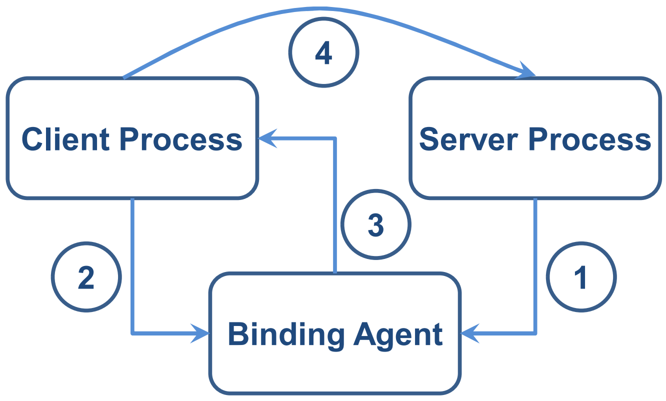
Binding Agent

- A server can also deregister with the binding agent when it is no longer prepared to offer service.
- The binding agent can also poll the servers periodically, automatically deregistering any server that fails to respond.

16 July 2024

91

Binding Agent Mechanism in RPC



1. Server registers with Binding Agent
2. Client request for Server’s location information
3. Binding agent returns Server’s location information to client
4. Client calls the Server

92

Binding Agent

- A client contacts the binding agent to locate a server.
- If the server is registered with the binding agent, it returns the handle of the server to the client.
- The binding agent's location is known to all nodes.

16 July 2024

93

Primitives for Binding Agent

- A binding agent interface usually involves three primitives:
  - *register* to be used by a server to register itself with the binding agent
  - *deregister* is again to be used by a server to deregister itself with the binding agent, and
  - *lookup* to be used by a client to locate a server.

16 July 2024

94

Advantages of using Binding Agent

- The binding agent mechanism supports multiple servers having the same interface type.
- Any of the available servers may be used to serve a client's call.
- This helps to achieve a degree of fault tolerance.

16 July 2024

95

Advantages of using Binding Agent

- When multiple servers provide the same service, the clients can be spread evenly to balance the load.
- This can be extended to allow servers to specify a list of users who may use a service.
- A binding agent would refuse to bind those clients to the servers who are not authorized to use a service.

16 July 2024

96

Limitations of using Binding Agent

- Overhead of binding clients to servers is high especially when the clients are short lived and service processes frequently migrate.
- A binding agent must be robust against failures and at the same time should not become a performance bottleneck.

16 July 2024

97

Limitations of using Binding Agent

- Distributing binding function among several binding agents with replicated information can satisfy both criteria.
- Unfortunately, replication typically involves extra overhead of keeping the multiple replicas consistent.

16 July 2024

98

When is it good to bind?

- Compilation Time Binding
  - the client and server modules are coded as if they were intended to be linked together.
  - e.g., server's network address is in client code
- Binding during Linking
  - server connects to binding agent during initialization process.
- Binding at Call Time
  - client binds to a server when it calls the server

16 July 2024

99

Compilation Time Binding

- The method is extremely inflexible
- If the server relocates or the interface changes, all client programs are to be revised and compiled again.
- The method is useful only in certain limited cases.
  - e.g., it may be used in an application whose configuration is expected to remain static for a fairly long time.

16 July 2024

100

Binding during Linking

- When binding agent gets a request message from the RPC Runtime of client then it binds the client and the server by returning to the client the server's handle
- Calls can take place once the client has received the server's handle.

16 July 2024

101

Binding during Linking

- The server's handle is cached by the client to avoid contacting the binding agent for subsequent calls to be made to the same server.
- Considering the overhead involved for binding agent, this method is more suitable for those situations in which a client calls a server several times once it is bound to it.

16 July 2024

102

Binding at Call Time

- A commonly used approach for binding at call time is the indirect call method.
- In this method, when a client calls a server for the first time, it passes the server's interface name and the arguments of the RPC call to the binding agent.

16 July 2024

103

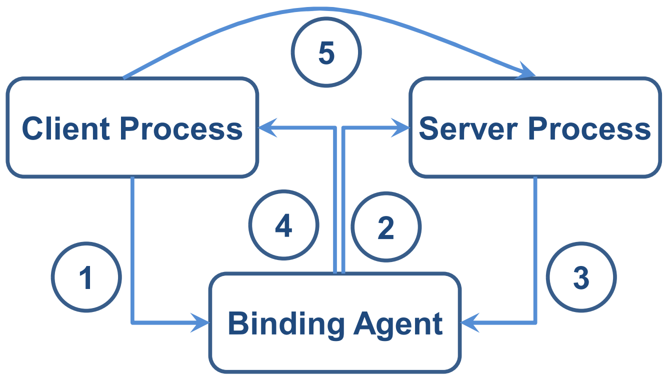
Binding at Call Time

- The binding agent finds the handle of the target server in its binding table.
- On behalf of the client, the binding agent sends the call to the server.
- When target server returns the results to the binding agent, the binding agent forwards the result and the handle both so that client can subsequently call the server directly.

16 July 2024

104

Binding at Call Time by Indirect Call



0. Server registers with Binding Agent
1. Client passes RPC request to Binding Agent
2. Binding Agent send RPC call request to Server
3. Server sends the result after execution to the Binding Agent
4. Binding agent returns result along with Server's handle to client
5. Subsequent calls are directly sent by Client to the Server

105

Can a Client/Server change binding?

- The flexibility to change existing bindings dynamically could be quite useful from a reliability point of view.
- As for example, a client may like to change its binding to another server
  - when a call to the already connected server fails or
  - when the quality of service goes below a critical threshold.

16 July 2024

106

Can a Client/Server change binding?

- Similarly, the server may want to connect the client to another server in situations such as
  - when the service needs to move to another node or
  - when the server decides to shut down for a maintenance.

16 July 2024

107

Multiple Simultaneous Binding

- In a system, a service may be exported by multiple servers.
- We have seen that, in general, a client is bound to a single server out of multiple servers of the same type.
- However, there may be situations when it is advantageous for a client to be bound simultaneously to all or multiple servers of the same type.

16 July 2024

108

Multiple Simultaneous Binding

- Logically, a binding like this supports multicast communication
- When a call is made, all the servers bound to the client for that service will receive and process the call.
  - e.g., a client may wish to update multiple copies of a file replicated at several nodes.
  - The client can be bound simultaneously to file servers of all those nodes where a replica of the file is located.

16 July 2024

109

Thanks for your kind attention

Questions??

110