**Remote Procedure Calls (RPC)**

-Rujuta Shah

(121044)

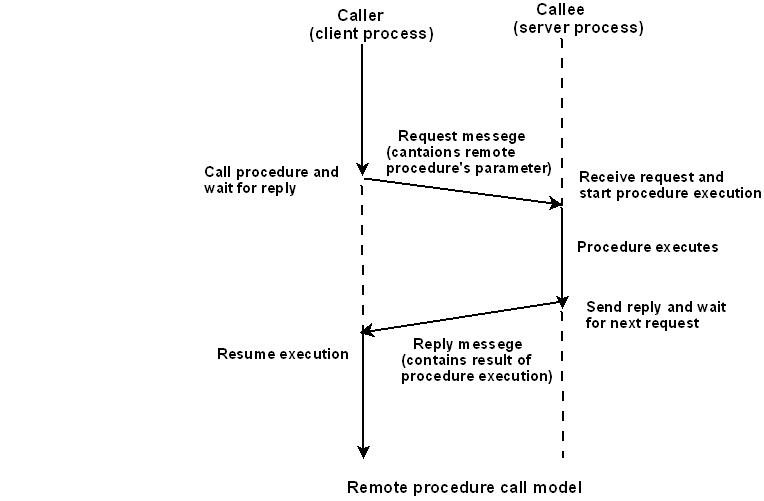
## Introduction:

* In 1984, Birrell and Nelson devised a mechanism to allow programs to call procedures on other machines.
* Remote Procedure Call (RPC) is an inter process communication technique to allow client and server software to communicate. RPC is a powerful technique for constructing distributed, client-server based applications. It is based on extending the notion of conventional or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them. By using RPC, programmers of distributed applications avoid the details of the interface with the network.
* RPC comes under the Application-Oriented Design, where the client-server communication is in the form of Procedure Calls. We call the machine making the procedure call as client and the machine executing the called procedure as server.

## What is a Stub?

* For every procedure being called there must exist a piece of code which knows which machine to contact for that procedure. Such a piece of code is called a Stub. On the client side, for every procedure being called we need a unique stub. However, the stub on the server side can be more general; only one stub can be used to handle more than one procedures. Also, two calls to the same procedure can be made using the same stub.

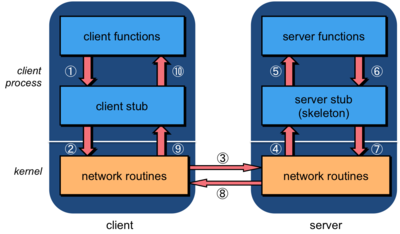
## Remote Procedure call model:



## On arrival of request message, the server processes the following tasks:

* + extracts the procedure’s parameters,
  + computes the result,
  + sends a reply message, and
  + Then awaits the next call message.

## Working of RPC



1. The client calls a local procedure, called the client stub. To the client process, it appears that this is the actual remote procedure, which it can call using a local procedure call (since the stub is a local procedure). The client stub packages the arguments to the remote procedure (this may involve converting them to a standard format) and builds one or more network messages. The packaging of arguments into a network message is called marshaling.
2. Network messages are sent by the client stub to the remote system (by writing to a socket via system calls to the local kernel).
3. Network messages are transferred by the kernel to the remote system via some protocol (either connectionless or connection-oriented).
4. A server stub process, sometimes called a skeleton, on the server receives the messages. It unmarshals the arguments from the messages and, if needed, converts them from a standard format into a machine-specific format.
5. The server stub calls a local procedure call to the actual server function, passing it the arguments that it received from the client. The server function gets the illustion that it was called locally by the client since the server stub calls it locally with the same parameters that the client transmitted.
6. When the server is finished, it returns to the server stub with its return values.
7. The server stub converts the return values to a standard format (if necessary) and marshals them into one or more network messages to send to the client stub.
8. Messages get sent back across the network to the client stub.
9. The client stub reads the messages from the local kernel.
10. It then returns the results to the client function (possibly converting them first). The client feels that it just received a return value from the remote function, unaware that all the network messaging took place.

The client code then continues its execution...

## Remote procedure calls vs. Local procedure calls:

Difference between remote procedure calls and local procedure calls:

* 1. Unlike local procedure calls, with remote procedure calls,
     1. Disjoint Address Space
     2. Absence of shared memory.
     3. Meaningless making call by reference, using addresses in arguments and pointers.
  2. RPC’s are more vulnerable to failure because of:
     1. Possibility of processor crashes or
     2. Communication problems of a network.

3. RPC’s are much more time consuming than LPC’s due to the involvement of communication network.

Due to these reasons, total semantic transparency is impossible.

## What happens when things go wrong?

* There are more opportunities for errors. A server can generate an error, there might be problems in the network, the server can crash, or the client can disappear while the server is running code for it.
* The transparency of remote procedure calls breaks here since local procedure calls have no concept of the failure of the procedure call.
* Because of this, programs using remote procedure calls have to be prepared to either test for the failure of a remote procedure call or catch an exception.

## Advantages of RPC:

* Server independent
* Process-oriented and thread oriented models supported by RPC
* The development of distributed systems is simple because it uses straightforward semantics and easier.
* Like the common communications between the portions of an application, the development of the procedures for the remote calls is quite general.
* The procedure calls preserves the business logics which is apt for the application.
* The code re-writing / re-developing effort is minimized.
* Enables the usage of the applications used in the distributed environment, not only in the local environment.

## Disadvantages of RPC:

* Context switching increases scheduling costs
* RPC is not a standard – it is an idea that can be implemented in many ways
* RPC does not solve the most of the distribution creation problems
* RPC is only interaction based. This does not offer any flexibility in terms of hardware architecture.

## References:

* <http://www.tldp.org/LDP/nag/node128.html>
* <https://technet.microsoft.com/enus/library/cc787851(v=ws.10).aspx>
* <http://www.cs.cf.ac.uk/Dave/C/node33.html>
* <http://www.cse.iitk.ac.in/users/dheeraj/cs425/lec26.html>