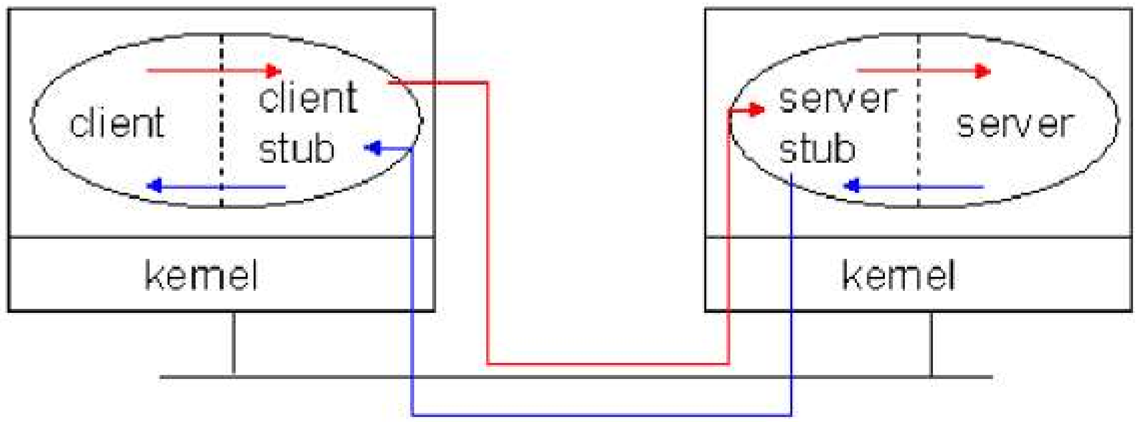
Remote Procedure Calls

Submitted By:

Ruchi Mittal

Remote Procedure Call

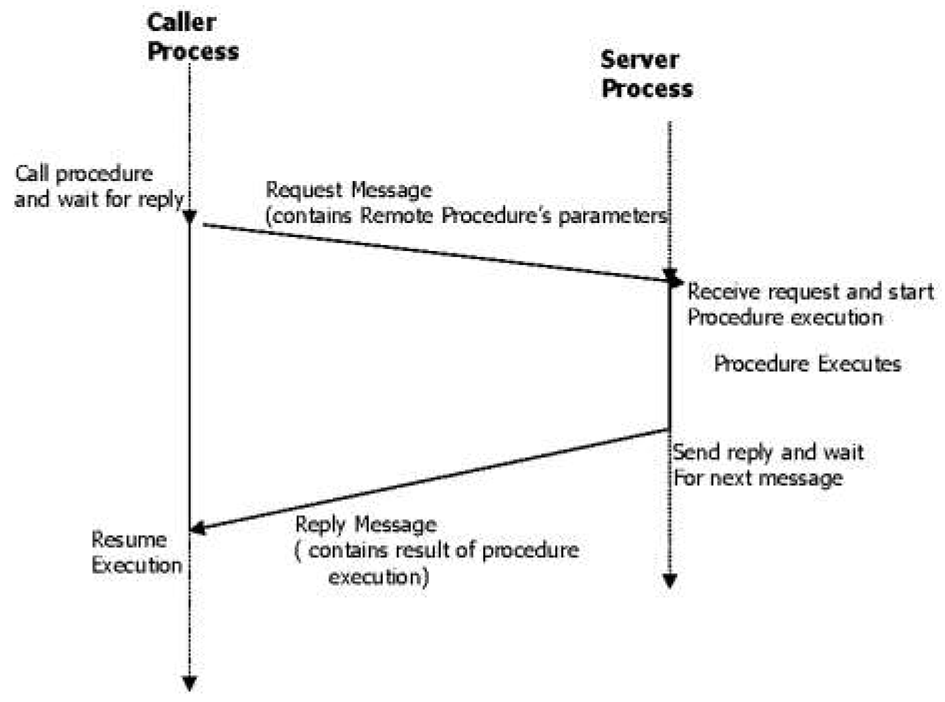
A convenient way to construct a client-server connection without explicitly writing send/ receive type programs (helps maintain transparency).



Remote Procedure Calls (RPC)

- General message passing model. Provides programmers with a familiar mechanism for building distributed applications/systems
- Familiar semantics (similar to LPC)
  - Simple syntax, well defined interface, ease of use, generality and IPC between processes on same/different machines.
- It is generally synchronous
- Can be made asynchronous by using multi-threading

A typical model for RPC


Caller Process
Server Process
Call procedure and wait for reply
Request Message (contains Remote Procedure's parameters
Receive request and start Procedure execution
Procedure Executes
Send reply and wait For next message
Resume Execution
Reply Message (contains result of procedure execution)

RPC continued...

- Transparency of RPC
  - Syntactic Transparency
  - Semantic Transparency

- Unfortunately achieving exactly the same semantics for RPCs and LPCs is close to impossible

  - Disjoint address spaces
  - More vulnerable to failure
  - Consume more time (mostly due to communication delays)

Implementing RPC Mechanism

- Uses the concept of stubs; A perfectly normal LPC abstraction by concealing from programs the interface to the underlying RPC
- Involves the following elements
  - The client
  - The client stub
  - The RPC runtime
  - The server stub
  - The server

Remote Procedure Call (cont.)

- Client procedure **calls** the client stub in a normal way
- Client stub **builds** a message and **traps** to the kernel
- Kernel **sends** the message to remote kernel
- Remote kernel **gives** the message to server stub
- Server stub **unpacks** parameters and **calls** the server
- Server **computes** results and **returns** it to server stub
- Server stub **packs** results in a message and **traps** to kernel
- Remote kernel **sends** message to client kernel
- Client kernel **gives** message to client stub
- Client stub **unpacks** results and **returns** to client

RPC servers and protocols…

- RPC Messages (call and reply messages)
- Server Implementation
  - Stateful servers
  - Stateless servers
- Communication Protocols
  - Request(R)Protocol
  - Request/Reply(RR) Protocol
  - Request/Reply/Ack(RRA) Protocol

RPC NG: DCOM & CORBA

- Object models allow services and functionality to be called from distinct processes
- DCOM/COM+(Win2000) and CORBA IIOP extend this to allow calling services and objects on different machines
- More OS features (authentication,resource management,process creation,...) are being moved to distributed objects.