Interprocess Communication - Messages

- Assume no explicit sharing of data elements in the address spaces of processes wishing to cooperate/communicate.
- Essence of message-passing is *copying* (although implementations may avoid actual copies whenever possible).
- Problem-solving with messages - has a feel of more active involvement by participants.

Hiding Message-Passing: RPC

The request/response communication is a basis for the *remote procedure call* (RPC) model.

- Think of a server as a module (data + methods).
- Think of a request message as a *call* to a server method.
  Each request carries an identifier for the desired method; the rest of the message contains the arguments.
- Think of the reply message as a *return* from a server method.
  Each reply carries an identifier for the matching call; the rest of the message contains the result.

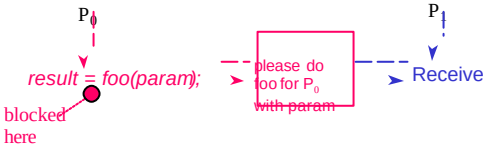
With a little extra glue, the messaging communication can be be hidden and made to look "just like a procedure call" to both the client and the server.

Remote Procedure Call - RPC

- *Looks* like a nice familiar procedure call

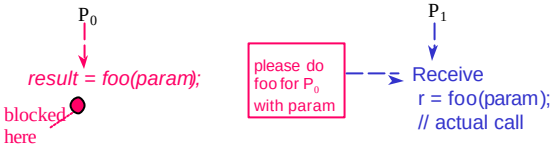
Remote Procedure Call - RPC

- *Looks* like a nice familiar procedure call



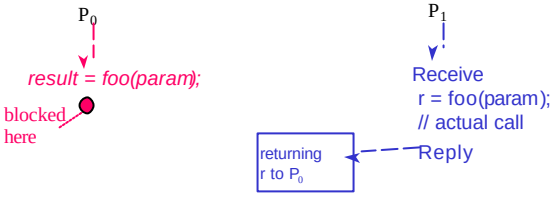
Remote Procedure Call - RPC

- *Looks* like a nice familiar procedure call



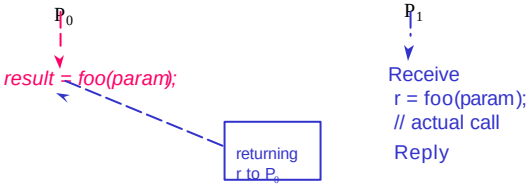
Remote Procedure Call - RPC

- *Looks* like a nice familiar procedure call



Remote Procedure Call - RPC

- *Looks* like a nice familiar procedure call



Remote Procedure Call - RPC

- *Looks* like a nice familiar procedure call

RPC Issues

1. RPC is a syntactically friendly communication/interaction model built *above* basic messaging or other IPC primitives.
   RPC is a nice model, but it is constrained and not fully transparent; not everyone likes it, and it more-or-less assumes threads.
2. Complex systems may be structured in the usual way as interacting modules, with processes imposing *protection boundaries* crossed using RPC.
   Interacting processes/modules may fail independently (?).
3. The RPC paradigm extends easily to distributed systems, but a variety of optimizations may be employed in the local cases.
   e.g., research systems and NT's *LPC* pass arguments in shared memory
4. The RPC model also extends naturally to object-based systems and object-based distributed systems.
   e.g., research systems, CORBA, Java *Remote Method Invocation*…there is an entire subculture out there

Rover
Joseph et al

- Relocatable dynamic objects (RDOs)
  - object with well defined interface that can be dynamically loaded by the client or server
- Queued Remote Procedure Calls (QRPCs)
  - a communication system which permits apps to continue non-blocking remote procedure call requests without a connection to a host

How does QRPC Work?

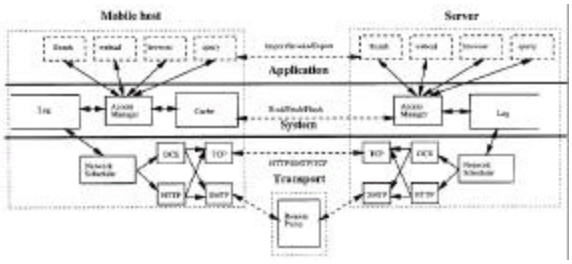
- QRPC maintains a log of requests
- the network scheduler makes an attempt to send the request to the server
- Rover delivers object if connected or if inexpensive
- log is maintained until reconnection or communication is less expensive (all requests sent)

QRPC Characteristics

- Simple message passing
- stub generation
- marshaling and unmarshaling of arguments
- at-most-once delivery semantics

~~When is an RPC no longer an RPC?~~

Rover Architecture



Access Manager

- Clients each run access manager
- handles all interactions between client apps and servers
- services requests for objects
- mediates network access
- manages object cache
- logs modifications to objects

Network Scheduler

- Responsible for processing log and forwarding QRPC to servers
- supports prioritization
- pre-orders transmission of QRPC's
- able to use several communications channels depending on cost and priority

Operational Log

- Log contains all "side-effecting" operations (server or client side) in the form of QRPC's
- Log is flushed back to server incrementally
- logs can be compacted and prioritized by applications

Object Cache

- Resides within applications address space
- offers several consistency controls
- updates to cache have different levels
  - tentative
  - permanent
- capable of pre-fetching

System Support Layer - Client Side

- Each client has its own separate address space to execute applications
- communication occurs through Local Remote Procedure Call (LRPC) with local Rover access manager
- access manager multi-threaded with non-preemptive servicing with cleanup background processes

Rover Interface

Function	Operation
Rover_AddWrite	Mutate an RDO
Rover_Export	Export a modified RDO
Rover_Flush	Flush a cached RDO
Rover_GetDV	Get an RDO's dependency vector
Rover_GetPid	Get an RDO's process ID
Rover_Import	Import an RDO
Rover_LoadApplication	Import an RDO and execute
Rover_MarkPermanent	Mark an operation permanent
Rover_NewSession	Create a new session
Rover_PendingWrites	Get a count of pending writes for an RDO
Rover_PromiseClaim	Claim a promise
Rover_QRPC	Issue a non-blocking QRPC
Rover_RPC	Issue a blocking RPC
Rover_Shutdown	Shut down a client application
Rover_Update	Update the Rover RDO cache from a local RDO

Table 1: Rover library functions.