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).
  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 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

P₀

result = foo(param);

Receive
r = foo(param);
// actual call

P₁

Please do
foo for P₀
with param

Blocked here

Remote Procedure Call - RPC

- Looks like a nice familiar procedure call

P₀

result = foo(param);

Receive
r = foo(param);
// actual call

P₁

Remote Procedure Call - RPC

- Looks like a nice familiar procedure call

P₀

result = foo(param);

Receive
r = foo(param);
// actual call

P₁

Remote Procedure Call - RPC

- Looks like a nice familiar procedure call

P₀

result = foo(param);

Receive
r = foo(param);
// actual call

P₁

Reply

returning
r to P₀

Remote Procedure Call - RPC

- Looks like a nice familiar procedure call

P₀

result = foo(param);

Receive
r = foo(param);
// actual call

P₁

Reply

returning
r to P₀
**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 none or less assumes thread.
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/40RPC 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, COBRA, 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

- 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

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

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rover GetRO</td>
<td>Return an ROO</td>
</tr>
<tr>
<td>Rover GetLRO</td>
<td>Return a local ROO</td>
</tr>
<tr>
<td>Rover Fetch</td>
<td>Fetch a cached ROO</td>
</tr>
<tr>
<td>Rover GetRV</td>
<td>Get an ROO's dependency vector</td>
</tr>
<tr>
<td>Rover GetROOut</td>
<td>Get an ROO's output RO</td>
</tr>
<tr>
<td>Rover LROApplication</td>
<td>Return an RDO with context</td>
</tr>
<tr>
<td>Rover MultiLROApplication</td>
<td>Issue an operation to an RDO</td>
</tr>
<tr>
<td>Rover Notify</td>
<td>Clean up context</td>
</tr>
<tr>
<td>Rover Print</td>
<td>Get a list of pending requests for an ROO</td>
</tr>
<tr>
<td>Rover PrintLRO</td>
<td>Output a RDO</td>
</tr>
<tr>
<td>Rover QRPC</td>
<td>Issue a remote QRPC</td>
</tr>
<tr>
<td>Rover RPC</td>
<td>Issue a blocking RPC</td>
</tr>
<tr>
<td>Rover Sleep</td>
<td>Issue a sleep operation</td>
</tr>
<tr>
<td>Rover Update</td>
<td>Update the Rover ROO cache from a local ROO</td>
</tr>
</tbody>
</table>

Table 1. Rover library functions.