Today: Remote Procedure Call

- Goal: Make distributed computing look like centralized computing
- Allow remote services to be called as procedures
- Issues
  - How to pass parameters
  - Binding
  - Semantics in face of errors

Conventional Procedure Call

- (a) Parameter passing in a local procedure call: the stack before the call to read.
  - Stack pointer
  - The stack while the called procedure is active.

(a) Main program's local variables
(b) Main program's local variables
- nbytes
- buf
- fd
- return address
- read's local variables
Parameter Passing

• Local procedure parameter passing
  – Call-by-value: integers, floats, chars
  – Call-by-reference: arrays, complex data structures
• Make a remote procedure call look like a local one through:
  – Stubs
  – Marshalling

PRC between a Client and a Server
Stubs

- Client makes local call to the client stub
- Server stub calls the procedure
- Stubs take care of packaging parameters/results and sending messages
  - Packaging parameters is called *marshalling*
- Stub compiler generates client and server stubs automatically from specs in an Interface Definition Language (IDL)
  - Simplifies programmer’s task

Steps of a Remote Procedure Call

1. Client procedure calls client stub in normal way
2. Client stub builds a message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS
9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client
Example of an RPC

Marshalling

- Marshalling: packing parameters/results into a message
- Problem: different machines have different data formats
  - Intel: little endian, SPARC: big endian
- Solution: use a standard representation
  - Example: external data representation (XDR)
- Problem: how do we pass pointers?
  - If it points to a well-defined data structure, pass a copy and the server stub calls the server with a pointer to the local copy
- What about data structures containing pointers?
  - Prohibit
  - Chase pointers over network
Binding

• How does a client locate a server?
  – Use binding
• Binding service maps procedure names to an IP address and a port number
• Server
  – When it starts up, register the services it is providing with the binding service
• Client
  – Ask the binding service about a particular procedure and it returns the corresponding IP address and port number

Failure Semantics

• *Client unable to locate server: return error*
• *Lost request messages: timeout mechanisms*
  – Resend the request upon timeout
• *Lost replies: timeout mechanisms*
  – Make request idempotent
  – Use sequence numbers
• *Server crashes: did crash occur before or after operation?*
  – At least once semantics (SUN RPC)
  – At most once
  – No guarantee
  – Exactly once: desirable but difficult to achieve
Server Crash

- A server in client-server communication.
  (a) The normal case.
  (b) Crash after execution.
  (c) Crash before execution.

Server Crash Example (1)

- Client requests server to print some text
- Assume server sends an ACK when it receives a request.
- Three events that can happen at the server:
  - Send the completion message (M)
  - Print the text (P)
  - Crash (C)
Server Crash Example (2)

The three events can occur in six different orderings:
1. M → P → C: A crash occurs after sending the completion message and printing the text.
2. M → C (→ P): A crash happens after sending the completion message, but before the text could be printed.
3. P → M → C: A crash occurs after sending the completion message and printing the text.
4. P → C (→ M): The text printed, after which a crash occurs before the completion message could be sent.
5. C (→ P → M): A crash happens before the server could do anything.
6. C (→ M → P): A crash happens before the server could do anything.

Server Crash Example (3)

<table>
<thead>
<tr>
<th>Client Reissue strategy</th>
<th>Client Strategy M → P</th>
<th>Server Strategy P → M</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MPC</td>
<td>MC(P)</td>
</tr>
<tr>
<td>Always</td>
<td>DUP</td>
<td>OK</td>
</tr>
<tr>
<td>Never</td>
<td>OK</td>
<td>ZERO</td>
</tr>
<tr>
<td>Only when ACKed</td>
<td>DUP</td>
<td>OK</td>
</tr>
<tr>
<td>Only when not ACKed</td>
<td>OK</td>
<td>ZERO</td>
</tr>
</tbody>
</table>

OK = Text is printed once
DUP = Text is printed twice
ZERO = Text is not printed at all
Failure Semantics

• *Client crash: what happens to the server computation?*
  – Referred to as an orphan
  – *Extermination: log at client stub and explicitly kill orphans*
    • Overhead of maintaining disk logs
  – *Reincarnation: Divide time into epochs between failures and delete computations from old epochs*
  – *Gentle reincarnation: upon a new epoch broadcast, try to locate owner first (delete only if no owner)*
  – *Expiration: give each RPC a fixed quantum T; explicitly request extensions*
    • If client waits T before rebooting, all orphans are gone.