IMPLEMENTING REMOTE PROCEDURE CALLS

PRESENTED BY
SARASWATHY SOMASUNDARAM
10/07/2004

Outline

- Introduction
- RPC Binding
- Grapevine
- Simple Call, Complicated Call
- Light weight RPC
- Sun RPC
Introduction

- Transfer of control and data across a communication network
- Special case of Interprocess Communication
- Issues facing implementation --- Semantics of call, addressing the callee, integration to current system, protocols for transfer, integrity and security of data

RPC Model

![Typical Model of RPC]

- Client
  - Call proc
  - Resume

- Server
  - Receive Req
  - Exec
  - Send Rep
  - Reply Messg

TYPICAL MODEL OF RPC
Example - Interprocess Communication

\[
\text{Count} = \text{read} (\text{Fd}, \text{Buf}, \text{nbytes})
\]

<table>
<thead>
<tr>
<th>Local Variables for main</th>
<th>Local Variables for main</th>
<th>Local Variables for main</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
<td>SP</td>
<td>SP</td>
</tr>
<tr>
<td>Bytes</td>
<td>Buf</td>
<td>Fd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ret addr</td>
</tr>
<tr>
<td>Read's local variables</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ret addr</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Before call to Read       Active Procedure       After return to Caller

Issues

- Indistinguishable local call and remote procedure calls
- Syntactic and Semantic Transparency
- Determining the location and identity of the callee
Components

- User
- User stub
- RPC Runtime
- Server
- Server stub

Implementation

**RPC IMPLEMENTATION**

<table>
<thead>
<tr>
<th>Client Machine</th>
<th>Server Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>Server</td>
</tr>
<tr>
<td>Return</td>
<td>Call</td>
</tr>
<tr>
<td>Call</td>
<td>Exec</td>
</tr>
<tr>
<td>Client stub</td>
<td>Serverstub</td>
</tr>
<tr>
<td>Unpack</td>
<td>Pack</td>
</tr>
<tr>
<td>Rrpcruntime</td>
<td>Rrpc Runtime</td>
</tr>
<tr>
<td>Receive</td>
<td>Wait</td>
</tr>
<tr>
<td>Send</td>
<td>Call packet</td>
</tr>
<tr>
<td>Result Packet</td>
<td></td>
</tr>
</tbody>
</table>
### Parameter Passing

<table>
<thead>
<tr>
<th>Client Machine</th>
<th>Server Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>N = \text{sum} (4,7)</td>
<td>Sum(i,j)</td>
</tr>
<tr>
<td>sum 4 7</td>
<td>int i,j;</td>
</tr>
<tr>
<td></td>
<td>{ return(i + j); }</td>
</tr>
<tr>
<td>Kernel</td>
<td>Kernel</td>
</tr>
<tr>
<td>Stubs</td>
<td></td>
</tr>
</tbody>
</table>

Computing sum(4,7) remotely;

---

### Client – Server Binding

- Interface – Type, Instance
- The interface and the services offered
- Created by user
- Locate a server - Broadcasting
- Binding agent
Grapevine

- Provides message delivery, access control and resource location
- Multiple computers communicating via the internet
- Maintains registration database - info about the users, services, machines etc
- Entry -> RName
- Replicated data

Grapevine Structure

- Grapevine Entries
  - Individuals
    - Network address
  - Groups
    - Member List
    - Group of RNames
Environment

- Cedar programming environment
- 24-bit virtual address, 80 megabyte hard disk
- Uniform access across the network
- Local Ethernet
- Powerful, convenient for building programs

Event Sequence

<table>
<thead>
<tr>
<th>Caller Machine</th>
<th>Grapevine</th>
<th>Callee Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impor t[a,b]</td>
<td>Rec bind [a,b]</td>
<td>Rec in tab</td>
</tr>
<tr>
<td>Ret</td>
<td>get cor</td>
<td>Exp [a,b]</td>
</tr>
<tr>
<td>X&lt;- F[y]</td>
<td>Trans</td>
<td>Exp [a,b]</td>
</tr>
<tr>
<td>F=&gt;3</td>
<td></td>
<td>Ret</td>
</tr>
</tbody>
</table>

**SEQUENCE OF EVENTS IN BINDING AND A SUBSEQUENT CALL**
Simple Calls

- Call packet -- Call identifier, data for the procedure along with the arguments
- Call identifier – Eliminates duplicate transfers
- Components of the call identifier – (machine relative identifier, sequence number) activity
- No initiation of new call until the result is obtained
- Result packet - Result and the same call identifier sent earlier

Simple Calls – Cont’d

- Callee machine – table of sequence numbers

PACKETS TRANSMITTED IN A SIMPLE CALL

User  RPC+ Stub  RPC+ Stub  Server
Complicated Calls

- Problems faced – packet loss, long waits
- Caller sends probe packets – should be acknowledged
- Notifies the crash, failure of server, exceptions etc
- Large arguments- sent in multiple packets, last packet requires acknowledgement
Processes

- Stack-Idle server processes
- No extra process creation
- Reverts to idle state when completed
- Packets – Process identifier
- Four process swaps

Exception Handling

- Called signals
- Checks for catch phrase
- If terminated by jump, caller is notified
- Procedure activations are unwound
- Security→End to end data encryption
- Grapevine used as authentication service
Operating Systems

- Monolithic kernel OS - Large kernel separated from user programs
- Microkernel - Small kernel with primitive operations for user level servers
- Microkernel - Programmed separately with own address space which forms a domain
- Simple, flexible

Lightweight RPC

- Cross domain communication - Between domains on same machine
- Cross machine - Domains in different machines
- Lightweight – Uses optimized cross domain
- Uses special threads scheduling mechanism, hands off scheduling
- Client -> server (arg, thread)
- Server -> validates and creates a call
- Dispatches thread, starts server
Lightweight RPC

- Fewer copies of data
- Common stack is used and accessible to both the client and server
- All procedures have a call stub in client’s domain and entry stub in server’s domain
- Uses shared data structures, caching the domains to idle processes
- Higher performance

Sun RPC

- Developed by Sun Microsystems
- UNIX RPC
- Interface Definition->Program number, version number, procedures, i/o parameters
- IDL complier -> rpcgen
- RPC accepts one arg and produces one result
- Local binding agent-portmapper
- Handles exceptions
Conclusions

- RPC makes distributed computing easy and efficient
- Similar to the local calls
- Reduced cost
- Better Performance
References