EECS 591 Winter 2003
Handout 4

• The Client-Server Model
  – Section 1.5 Tanenbaum
  – Handout in class

• RPC
  – Section 2.2 Tanenbaum
  – Section 7.3 Tanenbaum
The Client-Server Model

**Key Idea:** Structure a distributed system as a collection (or group) of cooperating processes, called servers, that offer services to users, called clients.

![Diagram of Client-Server Model]

- Examples?
The Client-Server Model

General interaction between a client and a server.
The general organization of an Internet search engine into three different layers
Multitiered Architectures (1)

Alternative client-server organizations (a) – (e).
An example of a server acting as a client.
Modern Architectures

An example of horizontal distribution of a Web service.
Remote Procedure Call

- First suggested by Birrell & Nelson in ‘84
- Remote ops in the guise of a procedural interface
- Perform ops on other machines as if they were local
- No I/O, no msg. visible to the programmer
An adapted reference model for networked communication.
Conventional Procedure Call

(a) Parameter passing in a local procedure call: the stack before the call to read
(b) The stack while the called procedure is active
Client and Server Stubs

Principle of RPC between a client and server program.
Steps of a Remote Procedure Call

1. Client procedure calls client stub in normal way
2. Client stub builds 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
Issues

• Why is RPC more than a syntactic change?
  – Exchanging messages between different processors (arch)
  – Little-endian vs. big-endian
  – Character representation
  – Sending pointers?

• What’s parameter marshalling?
  – Exchanging data structures between programs in different languages on two different architectures
  – Should the transport protocol be able to handle this issue?

• Dynamic binding

• RPC semantics in the presence of failures
Passing Value Parameters (1)

Steps involved in doing remote computation through RPC:

1. Client call to procedure
2. Stub builds message
3. Message is sent across the network
4. Server OS hands message to server stub
5. Stub unpacks message
6. Stub makes local call to "add"
Passing Value Parameters (2)

(a) Original message on the Pentium
(b) The message after receipt on the SPARC
(c) The message after being inverted. The little numbers in boxes indicate the address of each byte
Parameter Specification and Stub Generation

a) A procedure
b) The corresponding message.

foobar( char x; float y; int z[5] )
{
    ....
}

Note: IDL to support application development.
Two Extensions to RPC

• Doors: RPC for processes on the same machine

• Asynchronous RPC
Doors

The principle of using doors as IPC mechanism.
Asynchronous RPC (1)

a) The interconnection between client and server in a traditional RPC

b) The interaction using asynchronous RPC
Asynchronous RPC (2)

A client and server interacting through two asynchronous RPCs
Binding a Client to a Server

- Locate the server’s machine
- Locate the service on that machine
RPC Semantics in the Presence of Failures

• The client is unable to locate the server.

• The request message from the client to the server is lost.

• The server crashes after receiving a request.

• The reply message from the server to the client is lost.

• The client crashes after sending a request.
Lost Request Messages
Server Crashes (1)

A server in client-server communication
a) Normal case
b) Crash after execution
c) Crash before execution
Server Crashes (2)

Different combinations of client and server strategies in the presence of server crashes.

<table>
<thead>
<tr>
<th>Reissue strategy</th>
<th>Strategy M -&gt; P</th>
<th>Strategy P -&gt; 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>