

CSE 451: Operating Systems Autumn 2003

Lecture 16 RPC

Hank Levy
levy@cs.washington.edu
Allen Center 596

What's Interesting about RPC?

- RPC = Remote Procedure Call
 - the most common means for remote communication
 - used both by operating systems and applications
 - NFS is implemented as a set of RPCs
 - HTTP is essentially RPC
 - DCOM, CORBA, Java RMI, etc., are just RPC systems
- Someday you, too, will likely have to write an application that uses remote communications
 - you'll likely model your remote communications on RPC
- RPC is really, really simple under the covers

12/8/03

© 2003 Hank Levy

2

Client/Server Communication

- The prevalent model for structuring distributed computation is the client/server paradigm
 - a **server** is a program (or collection of programs) that provides a service to other programs
 - e.g., file server, name server, web server, ...
 - server may span multiple nodes (clusters)
 - often, nodes are called servers too
 - e.g., the web server runs on a Dell server computer
 - a **client** is a program that uses the service
 - the client first **binds** to the server
 - locates it, establishes a network connection to it
 - the client then sends **requests** (with data) to perform **actions**, and the server sends **responses** (with data)
 - e.g., web browser sends a "GET" request, server responds with a web page

12/8/03

© 2003 Hank Levy

3

Messages

- Initially, people hand-coded messages to send requests and responses
 - but, this quickly gets tiresome
 - need to worry about message format
 - have to pack and unpack data from messages
 - servers have to decode messages and dispatch to handlers
 - messages are often asynchronous
 - after sending one, what do you do until response comes back?
 - messages aren't a natural programming model
 - maybe we could encapsulate messaging behind some abstraction that the OS provides...
 - then, we could just invoke library routines
 - the library routines would send messages for us, and wait for responses to come back.
 - › hmm....

12/8/03

© 2003 Hank Levy

4

Procedure Calls

- Procedure calls are a natural way to structure multiple modules inside a single program
 - every language supports procedure calls
 - semantics are well-defined and understood
 - programmers are used to them
- Idea: have servers export a set of procedures that can be called by client programs
 - similar to library API, class definitions, etc.
- Clients do a local procedure call, as though they were directly linked with the server
 - under the covers, the procedure call is converted into a message exchange with the server

12/8/03

© 2003 Hank Levy

5

Remote Procedure Calls

- So...now we know the main idea: use procedure calls as the model for distributed (remote) communication
- But, there are a bunch of hard issues:
 - how do we make the "remote" part of RPC invisible to the programmer?
 - and is that a good idea?
 - what are the semantics of parameter passing?
 - what if we try to pass by reference?
 - how do we bind (locate/connect-to) servers?
 - how do we handle heterogeneity?
 - OS, language, architecture, ...
 - how do we make it go fast?

12/8/03

© 2003 Hank Levy

6

RPC model

- A server defines the service interface using an **interface definition language (IDL)**
 - the IDL specifies the names, parameters, and types for all client-callable server procedures
 - example: ASN.1 in the OSI reference model
 - example: Sun's XDR (external data representation)
- A "**stub compiler**" reads the IDL declarations and produces two stub procedures for each server procedure
 - the server programmer implements the service's procedures and links them with the **server-side stubs**
 - the client programmer implements the client program and links it with the **client-side stubs**
 - the stubs manage all of the details of remote communication between client and server

12/8/03

© 2003 Hank Levy

7

RPC Stubs

- A client-side stub is a procedure that looks to the client as if it were a callable server procedure
 - it has the same API as the server's implementation of the procedure
 - a client-side stub is just called a "stub" in Java RMI
- A server-side stub looks like a caller to the server
 - it looks like a hunk of code that invokes the server procedure
 - a server-side stub is called a "skeleton" or "skel" in Java RMI
- The client program thinks it's invoking the server
 - but it's calling into the client-side stub
- The server program thinks it's called by the client
 - but it's really called by the server-side stub
- The stubs send messages to each other to make the RPC happen transparently

12/8/03

© 2003 Hank Levy

8

RPC example



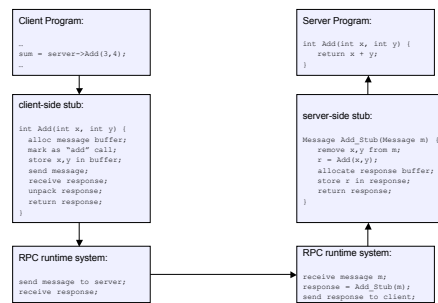
- If the server were just a library, then "Add" would just be a local procedure call

12/8/03

© 2003 Hank Levy

9

RPC example invocation



12/8/03

© 2003 Hank Levy

10

RPC Marshalling

- Marshalling is the packing of procedure parameters into a message packet
 - the RPC stubs call type-specific procedure to marshal or unmarshal the parameters of an RPC
 - the client stub marshals the parameters into a message
 - the server stub unmarshals the parameters and uses them to invoke the service's procedure
 - on return:
 - the server stub marshals the return value
 - the client stub unmarshals the return value, and returns them to the client program

12/8/03

© 2003 Hank Levy

11

RPC Binding

- Binding is the process of connecting the client to the server
 - the server, when it starts up, exports its interface
 - identifies itself to a network name server
 - tells RPC runtime that it is alive and ready to accept calls
 - the client, before issuing any calls, imports the server
 - RPC runtime uses the name server to find the location of the server and establish a connection
- The import and export operations are explicit in the server and client programs
 - a slight breakdown in transparency
 - more to come...

12/8/03

© 2003 Hank Levy

12

RPC Transparency

- One goal of RPC is to be as transparent as possible
 - make remote procedure calls look like local procedure calls
 - we've seen that binding breaks this transparency
- What else breaks transparency?
 - failures: remote nodes/networks can fail in more ways than with local procedure calls
 - network partition, server crash
 - need extra support to handle failures
 - server can fail independently from client
 - "partial failure": a big bugbear in distributed systems
 - if an RPC fails, was it invoked on the server?
 - performance: remote communication is inherently slower than local communication
 - if you're not aware you're doing a remote procedure call, your program might slow down an awful lot...

12/8/03

© 2003 Hank Levy

13

RPC and thread pools

- What happens if two client threads (or client programs) simultaneously invoke the same server procedure using RPC?
 - ideally, two separate threads will run on the server
 - so, the RPC run-time system on the server needs to spawn or dispatch threads into server-side stubs when messages arrive
 - is there a limit on the number of threads?
 - if so, does this change semantics?
 - if not, what if 1,000,000 clients simultaneously RPC into the same server?

12/8/03

© 2003 Hank Levy

14