Chapter 21
Remote Procedure Calls

Model 1) Communication-Oriented

Design the functional specifications of client/server and how they communicate. (sockets)

Model 2) Application-Oriented

Build the program in a procedural language (C) and put some procedures on a separate computer. (RPCs)

Call procedure on another machine and wait for the return.

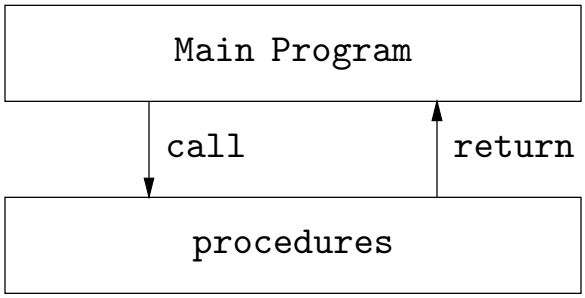
Program design/debugging advantages:

A comfortable (familiar) way to program.

You can build/test on a single machine as a normal program, then split.

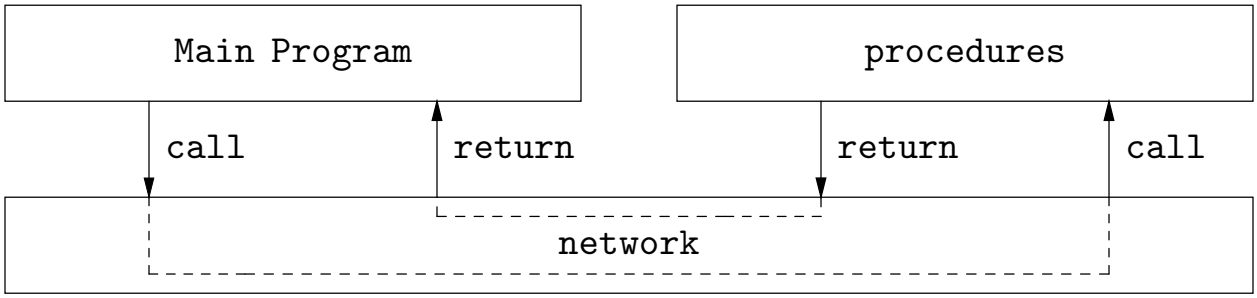
RPC vs Local

Local Calls



Parameters/results are passed using memory.
Must know where to parameters/results (stack)

Remote Calls



Parameters/results must passed over network

RPC Overview

1) Copy the parameters over.
2) Call the remote procedure.
3) Copy the answers back.
4) Return to the caller.

Calls are expensive,
once you jump, stay a while.

Call/copy uses UDP or TCP
(i.e. sockets)

Caller acts as client
Called procedure acts as server

You can't pass/return pointers,
you must copy the data.

A procedure uses a fixed number of parameters.

Detail: how to handle parameter and return values?
Solution: Allow only one parameter and result,
a single structure that contains everything.
XDR the parameter and result.

Remote Programs

A set of remote procedures
with shared global data

Calling standard procedures–by address
Relocatable: branch indirect (not direct)
Each procedure is assigned a number by the compiler
Call the procedure by number.
Look up address in an array.

Calling remote programs/procedures:
Assign each remote program a number.
Assign each procedure in a program a number.

Remote call:

1) machine
2) program number
3) procedure number
4) version number

Procedure 0: echo...is the remote program there

Version: allows multiple versions active at same time

RPC Semantics

RPC may select either TCP or UDP

TCP: Guarantees correctness of interaction.
Exactly one RPC call is made, exactly one set of parameters goes in each direction
or an error report is returned.

Operation has high overhead.

UDP: Will attempt the call.
If no response, RPC mechanism retries.
Retry parameters can be set by programmer.

If a response (return value) is received,
at least one call was made
(reply could get lost)

If no response is received,
the server could have received several calls

UDP principle: idempotent
build the client/server so duplicate calls don't cause problems

Lower overhead, but harder to program.

RPC Programming Support

Mutual exclusion: A remote program will accept only one active call at a time.

Portmapping:

Problem: TCP/UDP use 16-bit port numbers, RPC 32-bit numbers.
Cannot map RPC numbers onto ports directly.
Solution: RPC programs will obtain port numbers each time they execute.

Implementation: Each machine has an RPC port mapper that runs at a well-know port.
Portmapper: maintains a database of active port mappings for the RPC programs.
Call the portmapper with program number to get the programs port.
Connect to that port.

RPC Message Format

Call message: header, plus the parameter.

Return message: header, plus the result.

Each message is a single structure (header plus parameter/result).

The entire message must be XDR'd.

RPC knows the structure of the header
so it knows how to XDR the header,
but it must be told how to XDR the parameter/result.

Message content depends on structures used by particular RPC

The header includes:
Message id: used to match reply to call.
Message type: is this a call or reply
RPC version: so you can update RPC itself
Program, procedure and version number: specifying who is being called
Authentication: security and authorization stuff

Terminology

Marshaling Arguments

The arguments/results have to be XDR'd into the (liner/serial) buffer

This is called marshaling.

Receiver has to unmarshal the arguments/results

Marshaling and unmarshaling is done as part of XDRing and unXDRing the entire message (header plus arguments).

Authentication

Unix: senders machine, sending user, groups to which sending user belongs

You can use file system permissions to limit what users can do.

DES (encryption/decryption) is possible for secure channels.

If the receiver doesn't have the key he can't decrypt.

You can use encryption to authenticate the sender.
The sender signs the message,
the receiver can verify the signature.

Client/Server Model

The remote program is a server.
It runs on a "server" machine.
It receives RPC calls and returns answers.

The program making RPC calls is a client.

To make a call it must specify
1) which machine the remote program is on
2) which remote program on that machine
3) which procedure in the remote program.

Clients usually call one program, so:

RPC client will:
"connect" to a remote program on a machine
This connect creates a "handle".

A procedure call will use a "handle" to specify which program.

Getting TCP rpc handle opens a TCP connection to the remote program

RPC call: specify handle, procedure, and parameter.
RPC return: specify the result (return to caller).