Remote Procedure Calls

Client/Server Paradigm

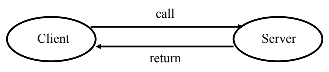
- Common model for structuring distributed computations
- A *server* is a program (or collection of programs) that
  - provides some *service*, e.g., file service, name service, …
  - may exist on one or more nodes.
- A *client* is a program that uses the service.
  - It first *binds* to the server,
    - i.e., locates it in the network and establishes a connection.
- The client then sends *requests* to perform actions;
  - Using messages to indicate the desired service and params.
  - The server returns a *response*.

Why not use messages?

- Although messages are flexible, they have problems:
  - Requires that programmer worry about message formats
  - Messages must be packed and unpacked
  - Server needs to decode messages to figure out the request
  - Messages are often asynchronous
  - They may require special error handling functions
- Basically using messages is not a convenient paradigm for most programmers.

Procedure Call

- More natural way is to communicate using procedure calls:
  - every language supports it
  - semantics are well defined and understood
  - natural for programmers to use
- Basic idea: define server as a module that *exports* a set of procedures callable by client programs.
- To use the server, the client just does a procedure call, as if it were linked with the server



(Remote) Procedure Call

- So, we would like to use procedure call as a model for distributed communication.
- Lots of issues:
  - how do we make this invisible to the programmer?
  - what are the semantics of parameter passing?
  - how is binding done (locating the server)?
  - how do we support heterogeneity (OS, arch., language)
  - etc.

Remote Procedure Call

- The basic model for Remote Procedure Call (RPC) was described by Birrell and Nelson in 1980, based on work done at Xerox PARC.
- Goal to make RPC as much like local PC as possible.
- Used computer/language support.
- There are 3 components on each side:
  - a user program (client or server)
  - a set of *stub* procedures
  - RPC runtime support

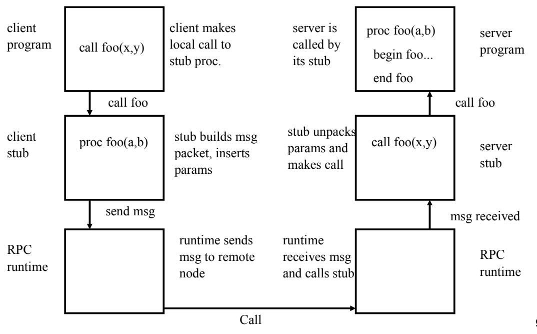
RPC

- Basic process for building a server:
  - Server program defines the server's interface using an *interface definition language* (IDL)
  - The IDL specifies the names, parameters, and types for all client-callable server procedures
  - A *stub compiler* reads the IDL and produces two stub procedures for each server procedure: a client-side stub and a server-side stub
  - The server writer writes the server and links it with the server-side stubs; the client writes her program and links it with the client-side stubs.
  - The stubs are responsible for managing all details of the remote communication between client and server.

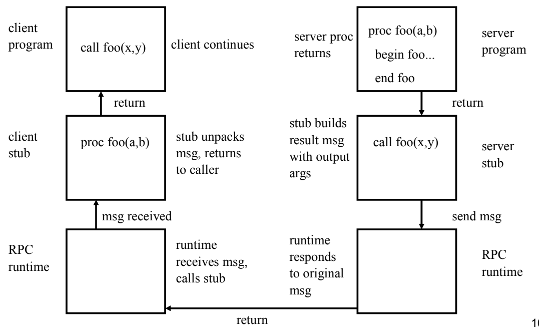
RPC Stubs

- Client-side stub is a procedure that looks to the client as if it were a callable server procedure.
- Server-side stub looks like a calling client to the server
- The client program thinks it is calling the server;
  - in fact, it's calling the client stub.
- The server program thinks it's called by the client;
  - in fact, it's called by the server stub.
- The stubs send messages to each other to make RPC happen.

RPC Call Structure


client program
call foo(x,y)
client makes local call to stub proc.
server is called by its stub
proc foo(a,b)
begin foo...
end foo
server program
call foo
call foo
client stub
proc foo(a,b)
stub builds msg packet, inserts params
stub unpacks params and makes call
call foo(x,y)
server stub
send msg
msg received
RPC runtime
runtime sends msg to remote node
runtime receives msg and calls stub
RPC runtime
Call

RPC Return Structure



RPC Binding

- Binding is the process of connecting the client and server
- The server, when it starts up, *exports* its interface,
  - identifying itself to a network name server and
  - telling the local runtime its dispatcher address.
- The client, before issuing any calls, *imports* the server,
  - which causes the RPC runtime to lookup the server through the name service and
  - contact the requested server to setup a connection.
- The *import* and *export* are explicit calls in the code.

RPC Marshalling

- Marshalling is packing of procedure params into message packet.
- RPC stubs call type-specific procedures to marshall (or unmarshall) all of the parameters to the call.
- On client side, client stub marshalls parameters into call packet;
  - On the server side the server stub unmarshalls the parameters to call the server's procedure.
- On return, server stub marshalls return parameters into return packet;
  - Client stub unmarshalls return params and returns to the client.

RPC Final

- RPC is most common model now for communications in distributed applications.
  - It is language support for distributed programming.
- RPC relies on a stub compiler to automatically produce client/server stubs from the IDL server description.
- RPC is commonly used, *even on a single node*, for communication between applications running in different address spaces. In fact, most RPCs are intra-node.