RPC basics

Lecture 6

SCS 3103 & IS 3008

Middleware Architecture

RPC Mechanisms

- Open Network Computing (ONC) – Sun Microsystems
- Distributed Computing Environment (DCE) – Open Software Foundation
- Both follow the same mechanism
- ISO RPC - IEEE

Remote Procedure Call

- How is a procedure called?
- How is an external procedure called?
- How is a remote procedure called?
  - Caller – client
  - Called – server
  - As if they were in the same machine

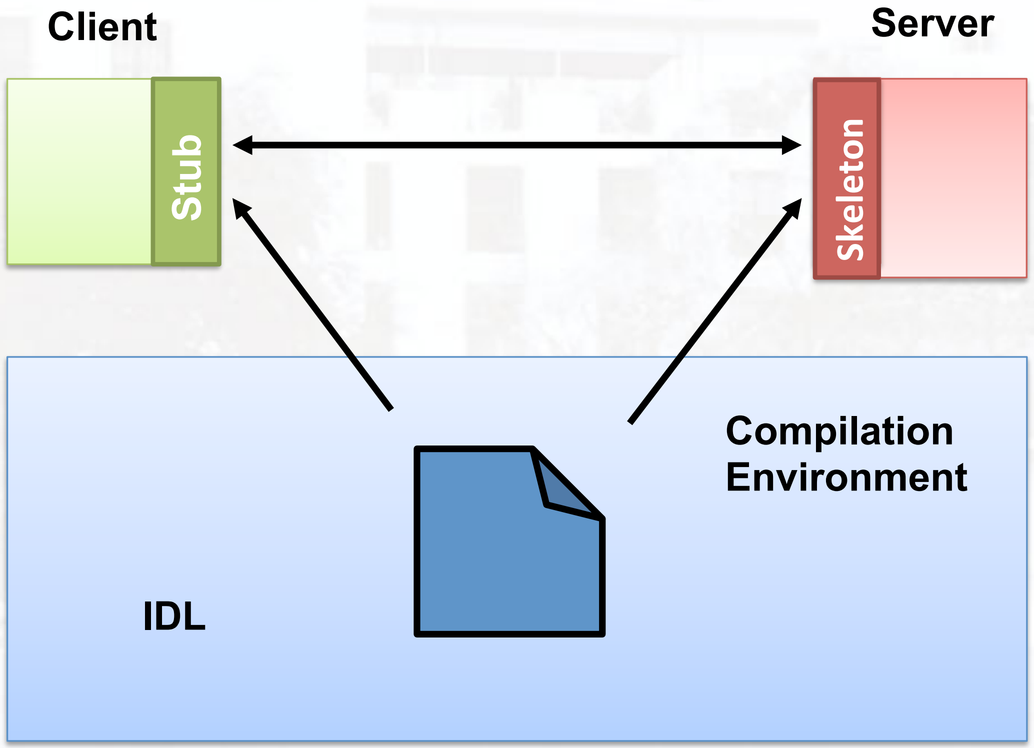
In RPC

- Instead of a header file we write a "Interface Definition Language (IDL)"
- Syntactically a IDL is similar to a header file but does something more.
- IDL generates client *stubs* and server *skeletons*.
- Small chunks of C code that are "Complied and Linked" to the client and server programs.

RPC

- RPC spans the Transport layer and the Application layer in the Open Systems Interconnection (OSI) model of network communication

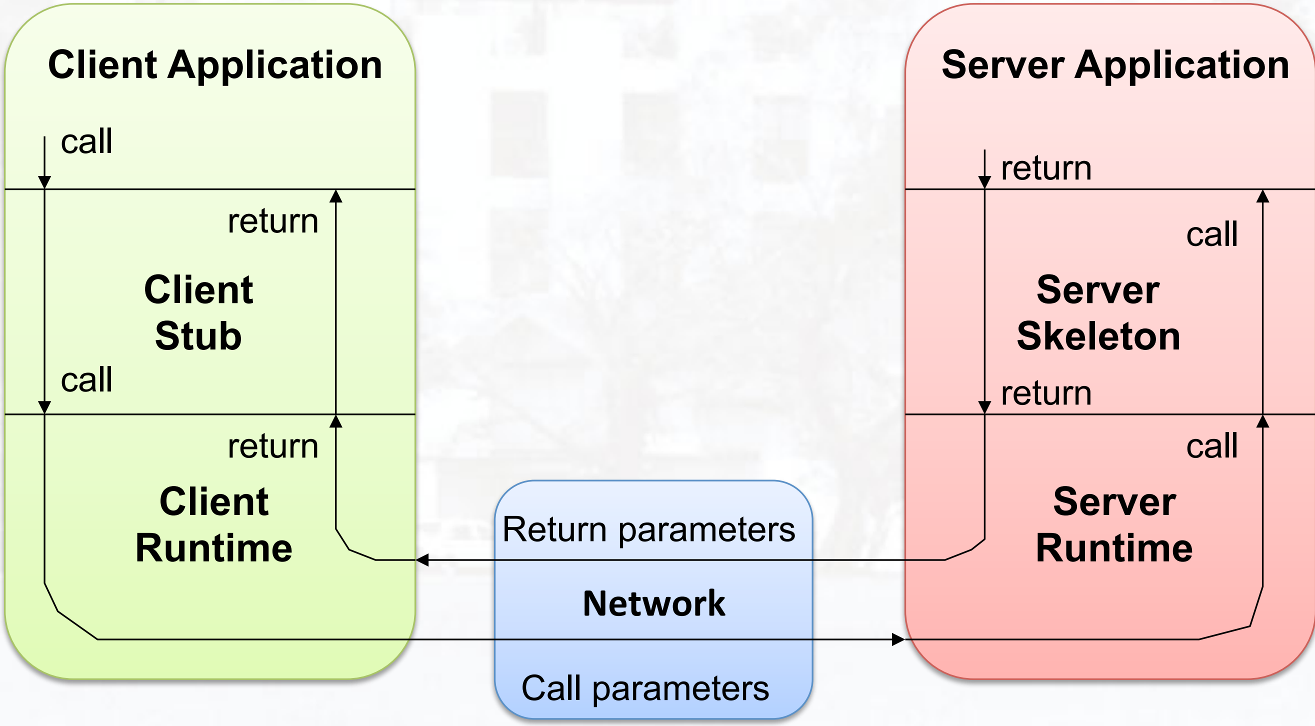
IDL



IDL

- Stub
  - Converts the parameters into a bit string
  - Send the message over the network
- Skeleton
  - Converts the message back into parameters
  - Calls the server procedure
- Converting parameters into message is marshalling

The RPC Model



Steps

1. The client calls the local stub procedure. Parameters are *marshalled*
2. Networking functions in the O/S kernel are called by the stub to send the message.
3. The kernel sends the message(s) to the remote system. This may be connection-oriented or connectionless.
4. A server skeleton unmarshals the arguments from the network message.
5. The server skeleton executes a local procedure call.

Steps

6. The procedure completes, returning execution to the server stub.
7. The server stub marshals the return values into a network message.
8. The return messages are sent back.
9. The client stub reads the messages using the network functions.
10. The message is unmarshalled. and the return values are set on the stack for the local process.

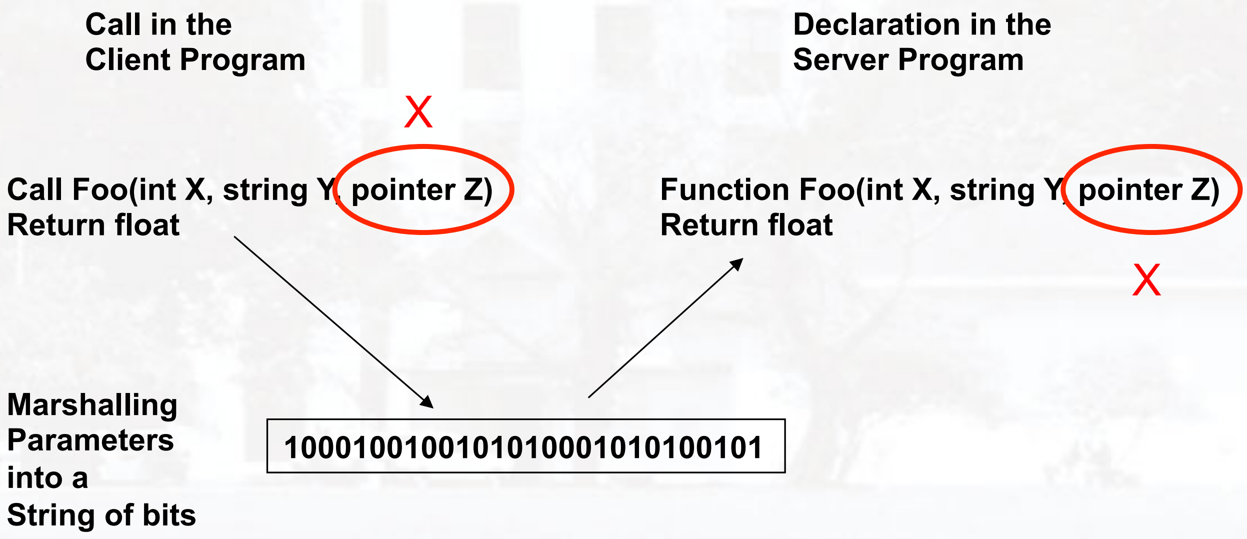
Generating Stubs and Skeletons

- Common RPC methods use implicit typing.
  - Both the server skeleton and the client stub must agree exactly on what the parameter types are for any remote call.
  - It must be done automatically.
- In RPC, generating the code is more complex than ordinary Procedure Calls.
  - The compiler must generate *separate* stubs and skeletons
  - These are embedded in the application

Generating Stubs and Skeletons

- The compiler must know which parameters are *in* parameters and which are *out*.
  - In parameters are sent from the client to server
  - Out parameters are sent back.
- Languages like C have no concept of *in* or *out* parameters. Therefore the compiler cannot be a standard C compiler, and the specification of the procedures cannot be done in C.

Marshalling



Call in the
Client Program

Declaration in the
Server Program

Call Foo(int X, string Y, pointer Z)
Return float

Function Foo(int X, string Y, pointer Z)
Return float

Marshalling
Parameters
into a
String of bits

100010010010101000101010010 1

Marshalling

- Advantage – Can handle different data formats
- Marshalling replaced by serialization
- Serialization - converting an object into a message
  - for storing on a disk
  - Sending over a network
- Serialization – converting parameters to a message

Errors in RPC

- Ordinary Procedure call
  - Divide by zero
  - Illegal instructions
  - Invalid memory reference
- In RPC
  - Can't find the server
  - Request to server is lost
  - Reply from server is lost
  - Server crashes
  - Client crashes

Weaknesses of RPC

- Lacks Multithreading
  - A client program is blocked when it' s calling a remote procedure
  - What if?
    - A message is lost in the network
    - If the server is slow
    - If the server hangs
- Partial solution is while the client is asking for data from the server it can read from the K/B or mouse
- The only way to achieve this is to use threads

Problems with RPC

- A similar problem occurs at the server end
  - Need a separate server thread for each client connection
  - A solution is to use a pool of server threads
  - Gets sophisticated
    - Transaction monitors
    - Synchronisation problems when threads share resources
      - Use locks & Semaphores

Problems in multithreading

- Finding bugs and testing
  - Each time a multithread program is run, the timing is slightly different
  - The actions in the thread are processed in a slightly different order
  - Bugs that depend on the order of processing are extremely hard to find
  - It is impossible to device a test plan