

Distributed Systems Lecture 5

Prof. Mehdi Jazayeri
TU Wien
jazayeri@tuwien.ac.at

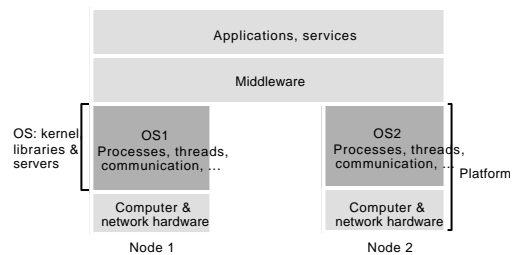
Lecture 5: Communication (2)

- Middleware protocols
 - Remote procedure call
 - Remote object invocation
- Message-oriented communication
- Stream communication

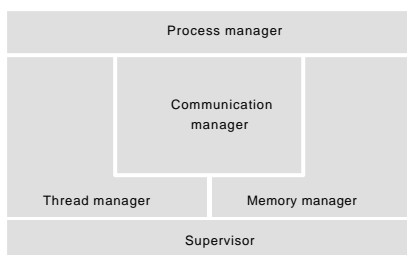
Middleware protocols

- The goal of the middleware communication protocols is to hide the existence of system boundaries. They provide high-level support for processes to communicate regardless of their location.
- Paradigms:
 - Procedure-based (RPC)
 - Object-based (RMI)
 - Message-based (MOM)
 - Continuous media

System layers



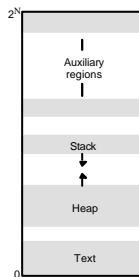
Core OS functionality



Remote procedure call

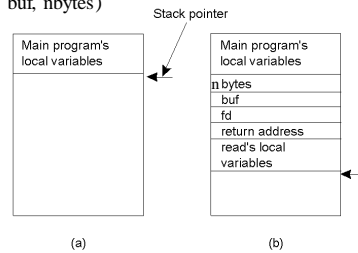
- Retain familiar procedural programming model
- Rely on system to bind calling and called procedure
- Use exception handling to deal with failures
- How to implement?
- How to use?

address space for a process



Conventional Procedure Call

read (fd, buf, nbytes)



- a) Parameter passing in a local procedure call: the stack before the call to read
- b) The stack while the called procedure is active

RPC implementation issues

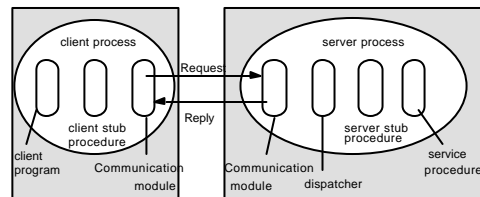
Client

- Program: whom to call?
- Kernel: what to do with data parameters?
- Kernel: where is server machine?
- Kernel: what to do with client program?

Server

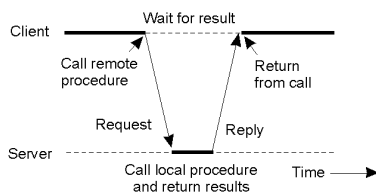
- Kernel: which procedure to invoke?
- Kernel: how to pass parameters?
- Kernel: what to do with result parameters?

Runtime structure of RPC



Stub procedures *marshal* and *unmarshal* parameters.
 Client stub marshals input parameters (and unmarshals return parameter)
 Server stub unmarshals input parameters and marshals return parameter

Client and Server Stubs

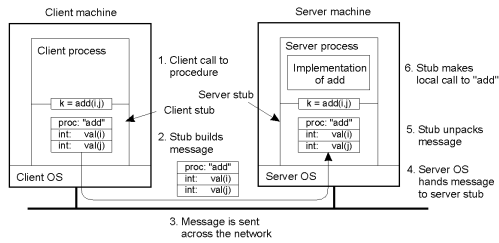


Principle of RPC between a client and server program.

Steps of a Remote Procedure Call

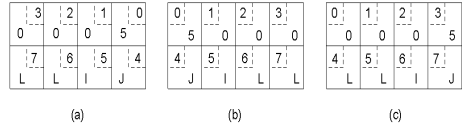
1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it 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

Passing Value Parameters (1)



Steps involved in doing remote computation through RPC

Passing Value Parameters (2)



- Original message on the Pentium
- The message after receipt on the SPARC
- The message after being inverted. The little numbers in boxes indicate the address of each byte

Parameter Specification and Stub Generation

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

(a)

foobar's local variables	
	x
	y
	5
	z[0]
	z[1]
	z[2]
	z[3]
	z[4]

(b)

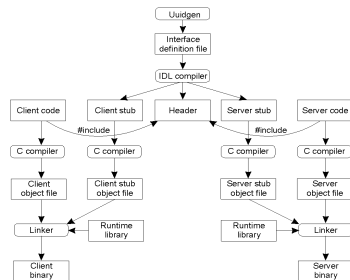
a) A procedure
b) The corresponding message.

Stubs can be automatically generated from interface description!
Marshaling = Serialization (in Java)

Interface Definition Language (IDL)

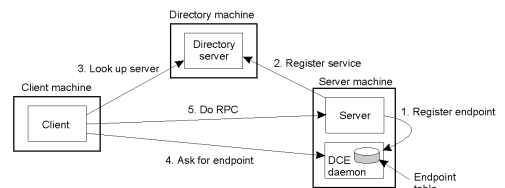
- An IDL supports the definition of procedures that may be called remotely
- It provides a set of "universal" types
- IDL *compiler* generates client and server *header files* and *stubs*
- Client code *includes* client header file and *links* to client stub
- Server code *includes* server header and *links* to server stub
- Caller and callee need not be in same language!

Writing a Client and a Server



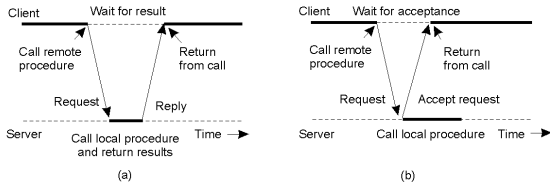
The steps in writing a client and a server in DCE RPC.

Binding a Client to a Server



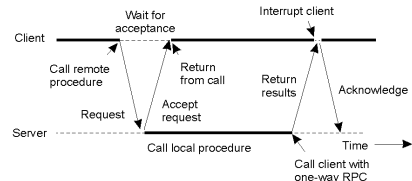
Client-to-server binding in DCE.

Asynchronous RPC (1)



- a) The interconnection between client and server in a traditional RPC
- b) The interaction using asynchronous RPC (when no reply needed, i.e. one-way RPC)

Asynchronous RPC (2)



A client and server interacting through two asynchronous RPCs

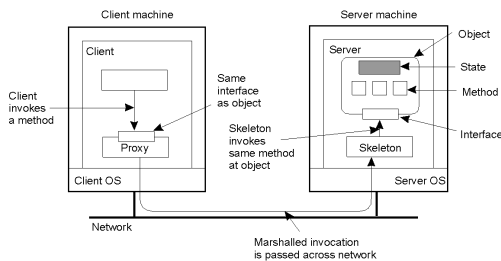
Summary of RPC

- RPC is an *end-to-end* service implemented on top of transport layers; it allows a *procedure* to be called in a different process
- An interface definition language supports the definition of interfaces and the generation of headers and stubs to be used by clients and servers
- RPC is supported by a directory service at runtime to register and locate remote procedures
- RPC is a one-to-one communication mechanism

Distributed objects

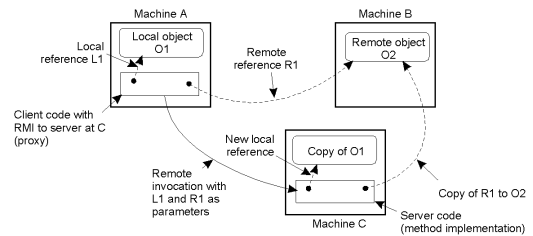
- Extend OO model for remote references and invocation (e.g. `obj.meth` where `obj` is remote)
- Objects reside around the network (not on a single system)
- Object references may refer to local and remote objects
- Calls may be made to methods of local or remote objects.

Distributed Objects



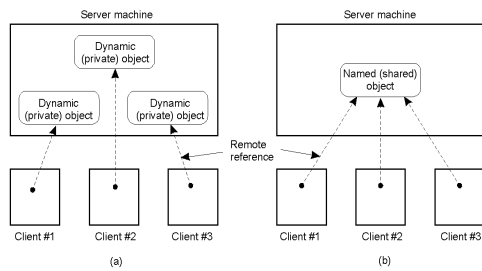
Common organization of a remote object with client-side proxy.

Parameter Passing



The situation when passing an object by reference or by value.

The DCE Distributed-Object Model



- a) Distributed dynamic objects in DCE.
- b) Distributed named objects

Java RMI

- Integrated into the language
 - *interface* construct used to define remote objects
 - *remote* interface indicates remotely callable object
 - Objects are passed between JVMs
 - Marshaling = serialization
- rmic = IDL compiler
- rmid = daemon listening for rmi incoming calls
- rmiregistry = directory service

Summary of Distributed Objects

- Distributed objects support invocation of remote objects transparently
- A globally unique "address" is needed as an object reference
- A "directory service" is needed to help find remote objects
- Parameters to remote methods may be passed by value or by reference