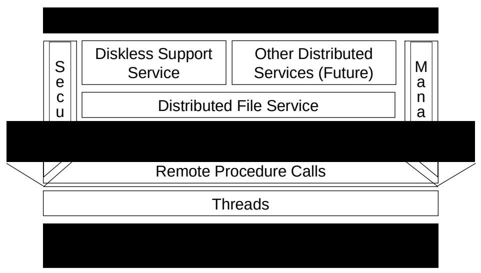
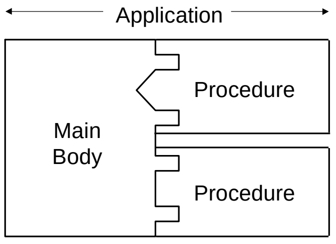
Remote Procedure Calls (RPC)

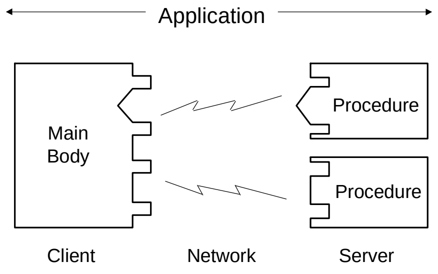
Architecture



Local Procedures



Remote Procedures



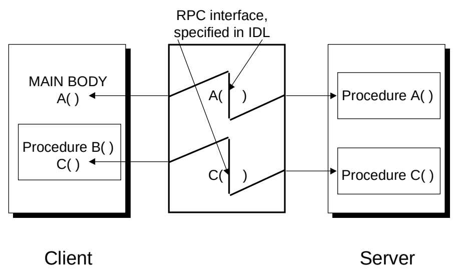
Why RPC?

- Extend familiar local procedure call paradigm
- Hide underlying networking technologies
- Mask differences in data representations
- A useful mechanism for distributing processing at a high level
  - Easier to use and more powerful than sockets

DCE RPC

- Provides interoperability for heterogeneous systems
- Works consistently with different types of transports
- Includes application development tools and runtime support
- Integrated with other DCE services:
  - Threads
  - Directory services
  - Security

Interfacing Local and Remote Procedures

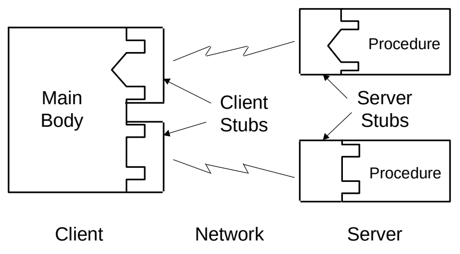


An Example Interface Definition in IDL

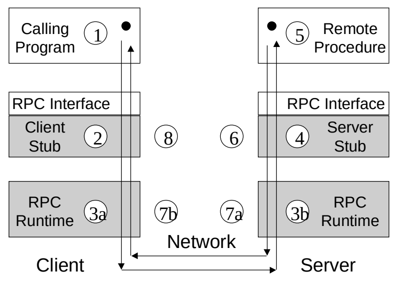
```
[ uuid(a01d0280-2d27-11c9-9fd3-08002b0ecef1),
   version(1.0) ]

interface math{
        const long ARRAY_SIZE = 10;
        typedef long array_type[ARRAY_SIZE];
        long get_sum([in] long first, [in] long second);
        void get_sums([in] array_type a,
                           [in] array_type b,
                           [out] array_type c);
        }
```

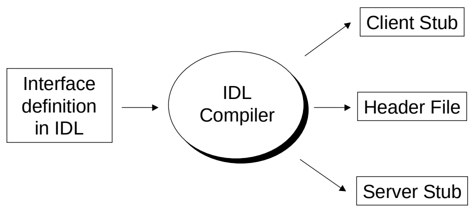
Stubs



Overview of RPC Operation



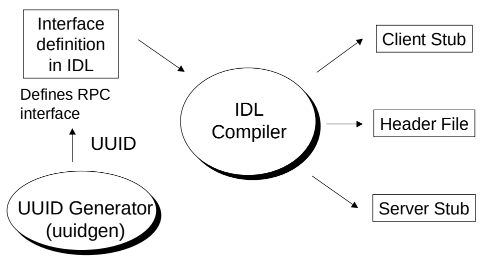
Where Stubs Come From



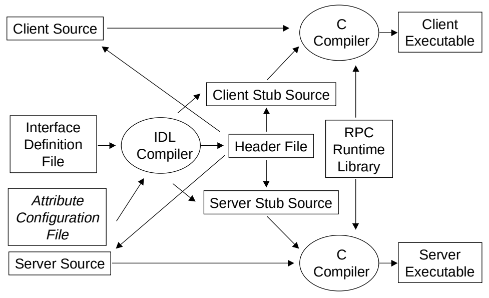
More on IDL

- IDL is a purely declarative language
  - Defines only types and procedure headers
- Its syntax is similar to C
- It supports:
  - Interface definition files (.idl)
  - Attribute configuration files (.acf)
- Familiar programming language data typing
  - Extensions for distributed programming are added

Interface Identification: UUIDs



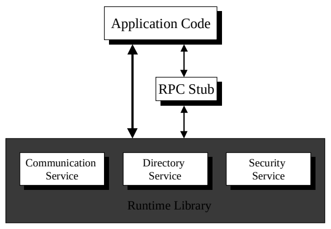
RPC Application Development: A Diagram



Requirements for Effective RPC

- Resolve differences in data representation
- Support a variety of execution semantics
- Support multi-threaded programming
- Provide good reliability
- Provide independence from transport protocols
- Ensure high degree of security
- Locate required services across networks

DCE RPC Runtime Library



Resolution of Data Representation Differences

- RPC automatically resolves data representation differences between heterogeneous systems
- Support is implemented in stubs generated by the IDL compiler
- DCE uses a receiver makes right scheme
- DCE's approach maximizes RPC performance between homogeneous systems

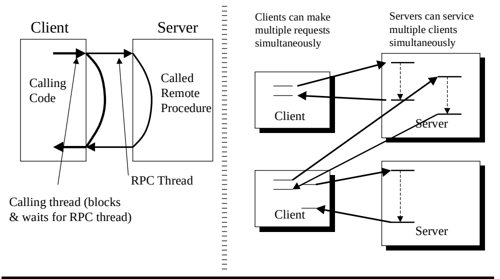
RPC Execution Semantics (1)

- If a request is sent, but no response is received, what should the requestor do?
  - If the request is blindly retransmitted, the remote procedure might be executed twice (or more)
  - If the request is not retransmitted, the remote procedure might not be executed at all
- Some remote procedures can safely be executed twice
  - Such procedures are said to be *idempotent*

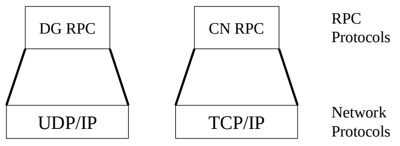
RPC Execution Semantics (2)

- Remote procedures must execute with desired behavior
- Execution semantics in DCE RPC:
  - At most once (Default)
  - Idempotent: at least once, possibly many times
  - Broadcast: a special case of idempotent semantics
  - Maybe: no response is expected, and the request might not get through, either

Integration of RPC with Threads


Client
Server
Calling Code
Called Remote Procedure
RPC Thread
Calling thread (blocks & waits for RPC thread)
Clients can make multiple requests simultaneously
Servers can service multiple clients simultaneously
Client
Server
Client
Server

DCE RPC Protocols



Specifying Protocols

- Client and server must specify a protocol sequence (called a protseq)
- A protseq contains:
  - RPC protocol
  - Network address family
  - Transport protocol
- Server has a choice with protocol sequences:
  - Support all available protocol sequences
  - Select the protocol sequence(s) to support

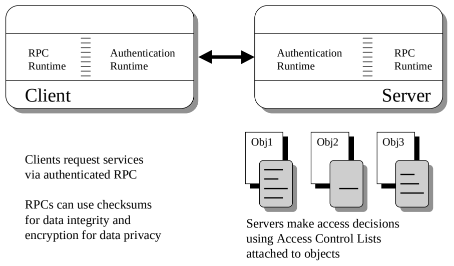
Daemons: rpcd and dced

- In DCE 1.0, a daemon called *rpcd* runs on every system that supports RPC servers
  - It stores transport endpoints (ports) in an endpoint map
  - Clients contact it to learn server endpoints
- In DCE 1.1, rpcd is replaced by *dced*
  - It performs the functions of rpcd
  - It improves the security of the endpoint map
  - It starts servers on demand

RPC Security (1)

- Distributed applications may require a number of security measures, including:
  - Authentication
  - Authorization (access control)
  - Data integrity
  - Data privacy
- DCE Security provides high level of security
- RPC is integrated with DCE Security

RPC Security (2)



Clients request services via authenticated RPC

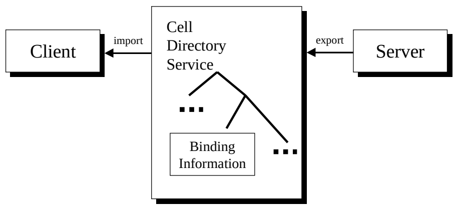
RPCs can use checksums for data integrity and encryption for data privacy

Servers make access decisions using Access Control Lists attached to objects

Location of Services (1)

- In a distributed environment
  - Servers need to advertise their services
  - Clients need to identify compatible servers
- The DCE Directory Service is used for this
- The RPC runtime can access the Directory Service
  - The Directory Service API used by RPC applications is called the Name Service Interface (NSI)

Location of Services (2)



Summary

- DCE RPC is a commercial-strength offering
- DCE RPC service provides:
  - Runtime facility
  - Development tools
- It is an integrated package
  - Integrated with directory service
  - Integrated with threads
  - Integrated with security
- A flexible tool for developers