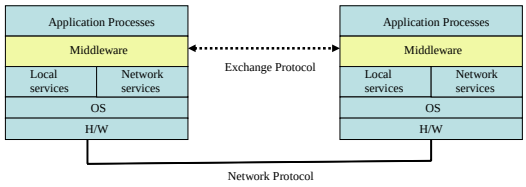
Middleware

- Definition: Distributed system services that have standard programming interfaces (APIs) and protocols and "sit in the middle" above OS and network software and below industry-specific applications.
- Functional View of Middleware
  - Information exchange services.
  - Management and support services needed for locating distributed resources and administering resources across the network.
  - Specialized services, e.g. transactional services and replication services for distributed databases, groupware services for collaborative applications, specialized services for multimedia applications.
  - Application-specific services..

Middleware



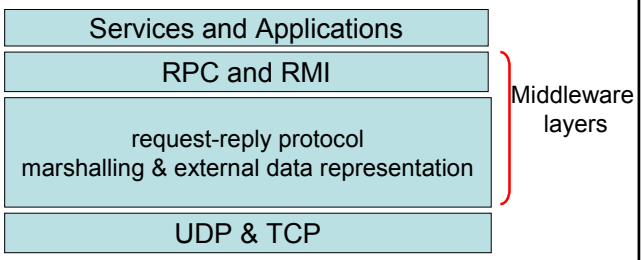
Middleware Technologies

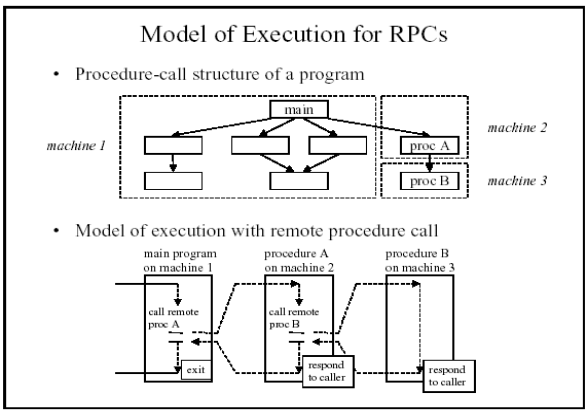
- Some of the technologies that are used in building middleware:
  - TCP/IP Sockets
  - RPC
  - RMI/CORBA
  - Web Services
  - Message-oriented Middleware (MOM)
- There are many middleware packages that have different architecture styles for different environments.

Process Interaction: RPC Model

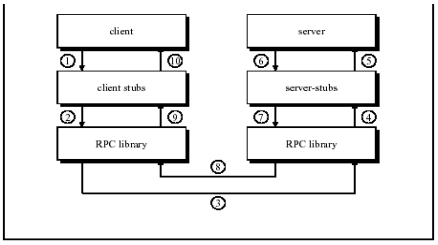
- Sockets API ≡ send & receive calls ≡ I/O
- Remote Procedure Calls (RPC): A higher level than sockets. RPC goals are:
  - to provide a procedural interface for distributed (i.e. remote) services
    - Looks like a local procedure.
  - to make distributed nature of service transparent to the programmer
    - No longer considered a good thing!

Middleware layers





Sequence of RPC Operation



- **Stubs:**
  - client-side proxy for the actual procedure on the server.
  - The *client-side stub* locates the server and marshals the parameters.
  - The *server-side stub* receives this message, un-marshaled the parameters, and performs the procedure on the server.

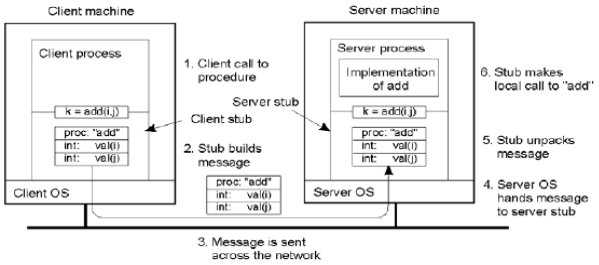
Steps of a Remote Procedure Call

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

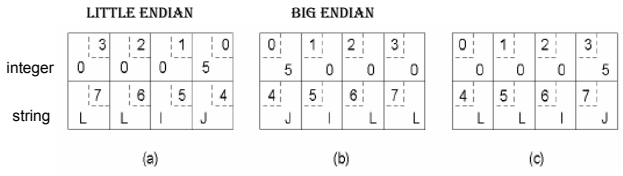
RPCs Issues

- Parameter Passing
  - call-by-value parameters
  - call-by-reference parameters
- Marshalling
  - atomic (simple) data structures
  - object (complex) data structures
- Exception handling
  - language dependent
  - asynchronous events

Parameter Passing in RPC (1)



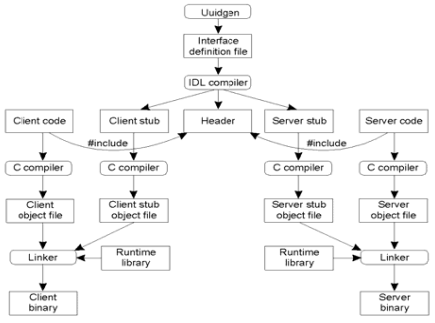
Parameter Passing in RPC (2)



- a) Original message on the Pentium
- b) The message after receipt on the SPARC
- c) The message after being inverted.

(The small numbers in boxes indicate the address of each byte)

Example
Writing a Client and a Server in DCE RPC



Remote Method Invocation: RMI

RMI = RPC + Object-orientation

- Java RMI
- Microsoft DCOM/COM+
- CORBA
  - Middleware that is language-independent
- SOAP
  - RMI on top of HTTP

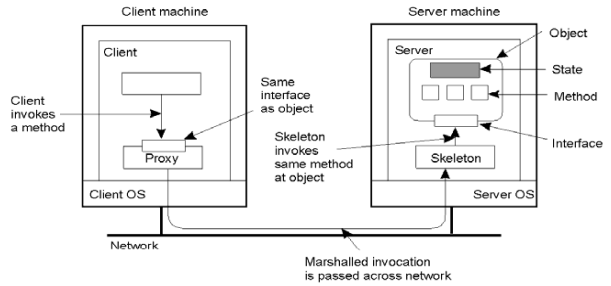
Object Model

- Object references
  - Objects accessed via object references
  - Object references can be assigned to variables, passed as arguments and returned as results
- Interfaces
  - Provides a signature of a set of methods (types of arguments, return values and exceptions) without specifying their implementations
- Actions (invocations)
- Exceptions
- Garbage Collection

Distributed Objects (1)

- Remote object references
  - An identifier that can be used throughout a distributed system to refer to a particular remote object
- Remote interfaces
  - CORBA provides an interface definition language (IDL) for specifying a remote interface
  - JAVA RMI: Java interface that extends the interface java.rmi.Remote.
- Actions: remote invocations: (Active and passive objects)
  - Active object = instantiated in a running process
  - Passive object = not currently active but can be made active
- Remote Exceptions: may arise for reasons such as partial failure or message loss.
  - In java RMI, each method of the interface declares java.rmi.RemoteException in its throws clause in addition to any application-specific clauses.
- Distributed Garbage Collection: cooperation between local garbage collectors needed

Distributed Objects(2)



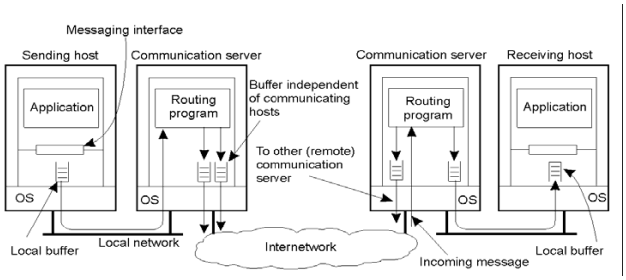
Issues in Implementing RMI

- Parameter Passing
  - Local & remote object references.
- Handling failures at client and/or server
  - Client unable to locate server
  - Request message lost
  - Reply message lost
  - Server crashes after receiving a request
  - Client crashes after sending a request
- Supporting persistent objects, object adapters, dynamic invocations.
- Location service.

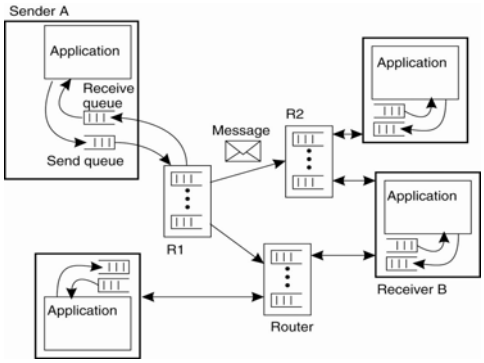
Message-Oriented Middleware (MOM)

- RPC and RMI (by default) support communication between two processes that are executing at the same time
  - transient communication
- Typically, the client is blocked until the RPC/RMI returns
  - synchronous communication
- Not suitable for middleware that integrates applications in widely dispersed and large-scale distributed systems
  - Message-oriented middleware

General MOM Architecture



Message-Queuing Model(1)



Message-Queuing Model(2)

Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block.
Notify	Install a handler to be called when a message is put into the specified queue.

Basic interface to a queue in a message-queuing system.

Message-queuing systems vs Email systems

- The architecture of message-queuing systems is very similar to that for email services. However,
  - email systems primarily provide support for end users.
  - message-queuing systems enable persistent communication between processes regardless of what the process is doing.
- leads to different requirements, e.g., guaranteed message delivery, message priorities, logging facilities, load balancing, fault tolerance, sequencing...
- Example of MQM is “Message Channel Agent” (MCA) component of the IBM’s WebSphere.