Distributed Systems

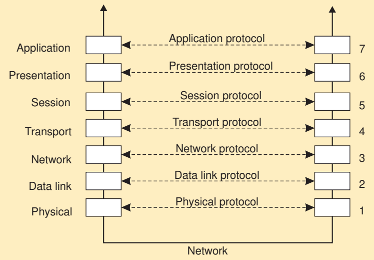
(3rd Edition)

Maarten van Steen Andrew S. Tanenbaum

Chapter 04: Communication

Edited by: Hicham G. Elmongui

Basic networking model



Drawbacks

- Focus on message-passing only
- Often unneeded or unwanted functionality
- Violates access transparency

Middleware layer

Observation

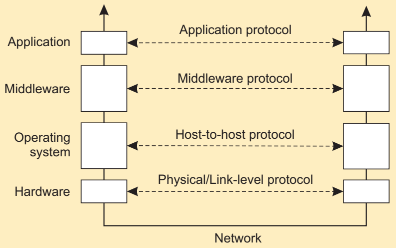
Middleware is invented to provide common services and protocols that can be used by many different applications

- A rich set of communication protocols
- (Un)marshaling of data, necessary for integrated systems
- Naming protocols, to allow easy sharing of resources
- Security protocols for secure communication
- Scaling mechanisms, such as for replication and caching

Note

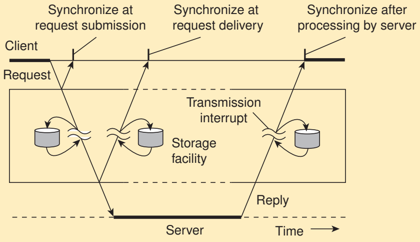
What remains are truly application-specific protocols... such as?

An adapted layering scheme



Types of communication

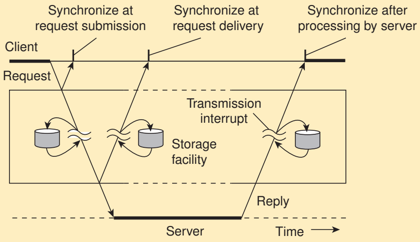
Distinguish...


Synchronize at request submission
Synchronize at request delivery
Synchronize after processing by server
Client
Request
Transmission interrupt
Storage facility
Reply
Server
Time

- Transient versus persistent communication
- Asynchronous versus synchronous communication

Types of communication

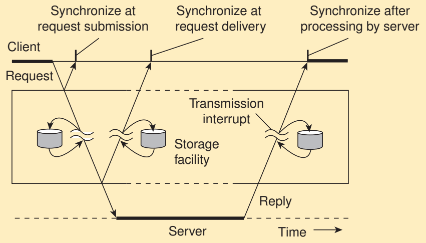
Transient versus persistent


Synchronize at request submission
Synchronize at request delivery
Synchronize after processing by server
Client
Request
Transmission interrupt
Storage facility
Reply
Server
Time

- Transient communication: Comm. server discards message when it cannot be delivered at the next server, or at the receiver.
- Persistent communication: A message is stored at a communication server as long as it takes to deliver it.

Types of communication

Places for synchronization


Synchronize at request submission
Synchronize at request delivery
Synchronize after processing by server
Client
Request
Transmission interrupt
Storage facility
Reply
Server
Time

- At request submission
- At request delivery
- After request processing

Client/Server

Some observations

Client/Server computing is generally based on a model of transient synchronous communication:

- Client and server have to be active at time of communication
- Client issues request and blocks until it receives reply
- Server essentially waits only for incoming requests, and subsequently processes them

Drawbacks synchronous communication

- Client cannot do any other work while waiting for reply
- Failures have to be handled immediately: the client is waiting
- The model may simply not be appropriate (mail, news)

Messaging

Message-oriented middleware

Aims at high-level persistent asynchronous communication:

- Processes send each other messages, which are queued
- Sender need not wait for immediate reply, but can do other things
- Middleware often ensures fault tolerance

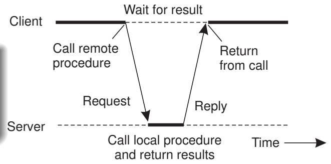
Basic RPC operation

Observations

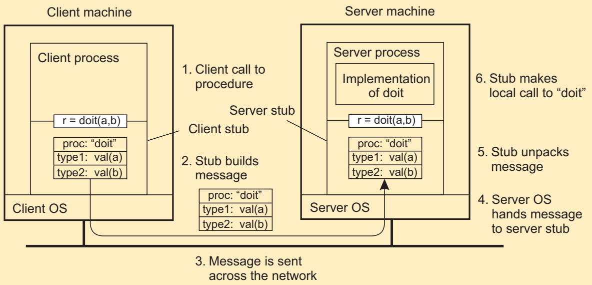
- Application developers are familiar with simple procedure model
- Well-engineered procedures operate in isolation (black box)
- There is no fundamental reason not to execute procedures on separate machine

Conclusion

Communication between caller & callee can be hidden by using procedure-call mechanism.



Basic RPC operation



1. Client procedure calls client stub.
2. Stub builds message; calls local OS.
3. OS sends message to remote OS.
4. Remote OS gives message to stub.
5. Stub unpacks parameters; calls server.
6. Server does local call; returns result to stub.
7. Stub builds message; calls OS.
8. OS sends message to client's OS.
9. Client's OS gives message to stub.
10. Client stub unpacks result; returns to client.

RPC: Parameter passing

There's more than just wrapping parameters into a message

- Client and server machines may have different data representations (think of byte ordering)
- Wrapping a parameter means transforming a value into a sequence of bytes
- Client and server have to agree on the same encoding:
- How are basic data values represented (integers, floats, characters)
- How are complex data values represented (arrays, unions)

Conclusion

Client and server need to properly interpret messages, transforming them into machine-dependent representations.

RPC: Parameter passing

Some assumptions

- Copy in/copy out semantics: while procedure is executed, nothing can be assumed about parameter values.
- All data that is to be operated on is passed by parameters. Excludes passing references to (global) data.

Conclusion

Full access transparency cannot be realized.

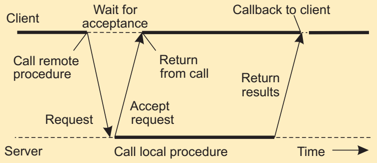
A remote reference mechanism enhances access transparency

- Remote reference offers unified access to remote data
- Remote references can be passed as parameter in RPCs
- Note: stubs can sometimes be used as such references

Asynchronous RPCs

Essence

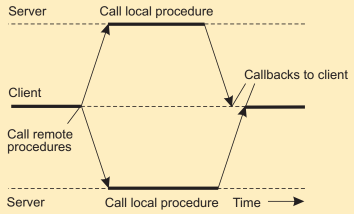
Try to get rid of the strict request-reply behavior, but let the client continue without waiting for an answer from the server.



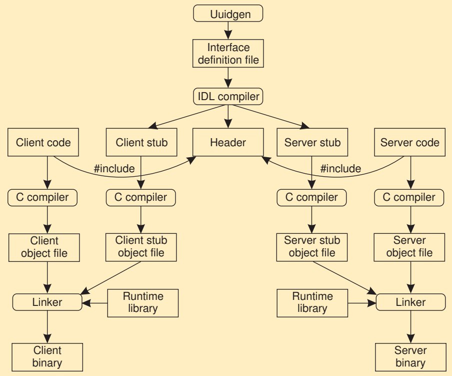
Sending out multiple RPCs

Essence

Sending an RPC request to a group of servers.



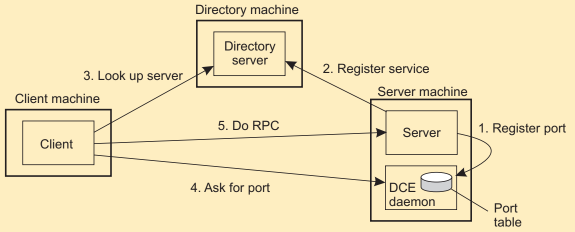
RPC in practice



Client-to-server binding (DCE)

Issues

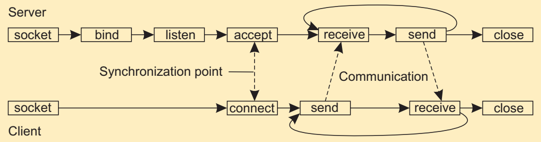
(1) Client must locate server machine, and (2) locate the server.



Transient messaging: sockets

Berkeley socket interface

Operation	Description
socket	Create a new communication end point
bind	Attach a local address to a socket
listen	Tell operating system what the maximum number of pending connection requests should be
accept	Block caller until a connection request arrives
connect	Actively attempt to establish a connection
send	Send some data over the connection
receive	Receive some data over the connection
close	Release the connection



Making sockets easier to work with

Observation

Sockets are rather low level and programming mistakes are easily made. However, the way that they are used is often the same (such as in a client-server setting).

Alternative: ZeroMQ

Provides a higher level of expression by pairing sockets: one for sending messages at process *P* and a corresponding one at process *Q* for receiving messages. All communication is asynchronous.

Three patterns

- Request-reply
- Publish-subscribe
- Pipeline

MPI: When lots of flexibility is needed

Sockets deemed insufficient

- They were at wrong level of abstraction by supporting only simple *send* and *receive* operations
- They were not considered suitable for the proprietary protocols developed for high-speed interconnection networks

Message-Passing Interface (MPI)

- a standard for message passing that is hardware and platform independent
- is designed for parallel applications and as such is tailored to transient communication
- makes direct use of the underlying network
- assumes that serious failures such as process crashes or network partitions are fatal and do not require automatic recovery

Message-oriented middleware

Essence

Asynchronous persistent communication through support of middleware-level queues. Queues correspond to buffers at communication servers.

Operations

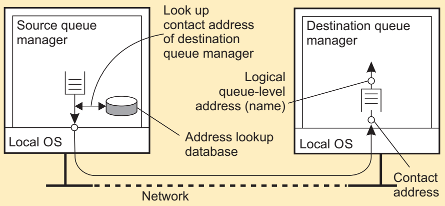
Operation	Description
`put`	Append a message to a specified queue
`get`	Block until the specified queue is nonempty, and remove the first message
`poll`	Check a specified queue for messages, and remove the first. Never block
`notify`	Install a handler to be called when a message is put into the specified queue

General model

Queue managers

Queues are managed by queue managers. An application can put messages only into a local queue. Getting a message is possible by extracting it from a local queue only ⇒ queue managers need to route messages.

Routing


Source queue manager
Look up contact address of destination queue manager
Destination queue manager
Logical queue-level address (name)
Address lookup database
Local OS
Local OS
Contact address
Network

Message broker

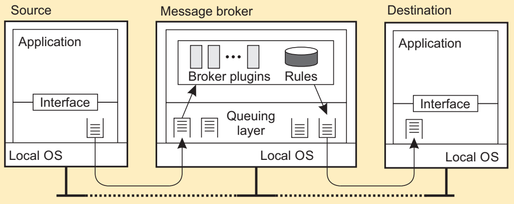
Observation

Message queuing systems assume a common messaging protocol: all applications agree on message format (i.e., structure and data representation)

Broker handles application heterogeneity in an MQ system

- Transforms incoming messages to target format
- Very often acts as an application gateway
- May provide subject-based routing capabilities (i.e., publish-subscribe capabilities)

Message broker: general architecture



Application-level multicasting

Essence

Organize nodes of a distributed system into an overlay network and use that network to disseminate data:

- Oftentimes a tree, leading to unique paths
- Alternatively, also mesh networks, requiring a form of routing

Flooding

Multicasting as Broadcasting

Construct an overlay network *per multicast group*.

Performance

A node belonging to several groups, will, in principle, need to maintain a separate list of its neighbors for each group of which it is a member.

Essence

P simply sends a message m to each of its neighbors. Each neighbor will forward that message, except to P, and only if it had not seen m before.

Performance

The more edges, the more expensive!