Chapter 4
Communication

Figure 4-1. Layers, interfaces, and protocols in the OSI model.

Layered Protocols (2)

Layered Protocols (1)

Middleware Protocols

Figure 4-3. An adapted reference model for networked communication.
Types of Communication

Synchronize at request submission
Synchronize at request delivery
Synchronize after processing by server

Client
Request

Transmission interrupt
Storage facility

Server
Reply

Time

Figure 4-4. Viewing middleware as an intermediate (distributed) service in application-level communication.

Conventional Procedure Call

Main program's local variables

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

Client and Server Stubs

Client
Call remote procedure
Wait for result

Server
Call local procedure and return results

Figure 4-6. Principle of RPC between a client and server program.

Remote Procedure Calls (1)

A remote procedure call occurs in the following steps:
1. The client procedure calls the client stub in the normal way.
2. The client stub builds a message and calls the local operating system.
3. The client's OS sends the message to the remote OS.
4. The remote OS gives the message to the server stub.
5. The server stub unpacks the parameters and calls the server.

Continued …
Remote Procedure Calls (2)

A remote procedure call occurs in the following steps (continued):
6. The server does the work and returns the result to the stub.
7. The server stub packs it in a message and calls its local OS.
8. The server's OS sends the message to the client's OS.
9. The client's OS gives the message to the client stub.
10. The stub unpacks the result and returns to the client.

Passing Value Parameters (2)

<table>
<thead>
<tr>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>L</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

(a)

Figure 4-8. (a) The original message on the Pentium.

Passing Value Parameters (3)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>J</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
</tbody>
</table>

(b)

Figure 4-8. (b) The message after receipt on the SPARC.
Passing Value Parameters (4)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>I</td>
<td>J</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4-8. (c) The message after being inverted. The little numbers in boxes indicate the address of each byte.

Parameter Specification and Stub Generation

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

(b)

Asynchronous RPC (1)

(a)

Asynchronous RPC (2)

(b)

Figure 4-10. (a) The interaction between client and server in a traditional RPC.

Figure 4-10. (b) The interaction using asynchronous RPC.
Asynchronous RPC (3)

Client
- Wait for acceptance
- Call remote procedure
- Request

Server
- Call local procedure
- Accept request
- Return from call
- Call client with one-way RPC
- Time
- Return results
- Acknowledge

Interrupt client

Figure 4-11. A client and server interacting through two asynchronous RPCs.

Writing a Client and a Server (1)

Client code
- Client stub
- Client stub object file
- Client stub library

Server code
- Server stub
- Server stub object file
- Server stub library

Figure 4-12. The steps in writing a client and a server in DCE RPC.

Writing a Client and a Server (2)

Three files output by the IDL compiler:

- A header file (e.g., interface.h, in C terms).
- The client stub.
- The server stub.

Binding a Client to a Server (1)

- Registration of a server makes it possible for a client to locate the server and bind to it.
- Server location is done in two steps:
  1. Locate the server’s machine.
  2. Locate the server on that machine.
Binding a Client to a Server (2)

Figure 4-13. Client-to-server binding in DCE.

Berkeley Sockets (1)

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket</td>
<td>Create a new communication end point</td>
</tr>
<tr>
<td>Bind</td>
<td>Attach a local address to a socket</td>
</tr>
<tr>
<td>Listen</td>
<td>Announce willingness to accept connections</td>
</tr>
<tr>
<td>Accept</td>
<td>Block caller until a connection request arrives</td>
</tr>
<tr>
<td>Connect</td>
<td>Actively attempt to establish a connection</td>
</tr>
<tr>
<td>Send</td>
<td>Send some data over the connection</td>
</tr>
<tr>
<td>Receive</td>
<td>Receive some data over the connection</td>
</tr>
<tr>
<td>Close</td>
<td>Release the connection</td>
</tr>
</tbody>
</table>

Figure 4-14. The socket primitives for TCP/IP.

Berkeley Sockets (2)

Figure 4-15. Connection-oriented communication pattern using sockets.

The Message-Passing Interface

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_send</td>
<td>Append outgoing message to a local send buffer</td>
</tr>
<tr>
<td>MPI_send</td>
<td>Send a message and wait until copied to local or remote buffer</td>
</tr>
<tr>
<td>MPI_send</td>
<td>Send a message and wait until receipt starts</td>
</tr>
<tr>
<td>MPI_send</td>
<td>Send a message and wait for reply</td>
</tr>
<tr>
<td>MPI_send</td>
<td>Pass reference to outgoing message, and continue</td>
</tr>
<tr>
<td>MPI_send</td>
<td>Pass reference to outgoing message, and wait until receipt starts</td>
</tr>
<tr>
<td>MPI_send</td>
<td>Receive a message, block if there is none</td>
</tr>
<tr>
<td>MPI_send</td>
<td>Check if there is an incoming message, but do not block</td>
</tr>
</tbody>
</table>

Figure 4-16. Some of the most intuitive message-passing primitives of MPI.
Message-Queuing Model (1)

Sender running
 Sender running
 Sender passive
 Sender passive

Receiver running
 Receiver passive
 Receiver running
 Receiver passive

Figure 4-17. Four combinations for loosely-coupled communications using queues.

Message-Queuing Model (2)

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put</td>
<td>Append a message to a specified queue</td>
</tr>
<tr>
<td>Get</td>
<td>Block until the specified queue is nonempty, and remove the first message</td>
</tr>
<tr>
<td>Poll</td>
<td>Check a specified queue for messages, and remove the first. Never block</td>
</tr>
<tr>
<td>Notify</td>
<td>Install a handler to be called when a message is put into the specified queue</td>
</tr>
</tbody>
</table>

Figure 4-18. Basic interface to a queue in a message-queuing system.

General Architecture of a Message-Queuing System (1)

Sender

Look up transport-level address of queue

Receiver

Queueing layer

Address look-up database

Local OS

Transport-level address

Network

Figure 4-19. The relationship between queue-level addressing and network-level addressing.

General Architecture of a Message-Queuing System (2)

Sender

Application

Receive queue

Send queue

Message

R1

Router

R2

Receivers

Application

Figure 4-20. The general organization of a message-queuing system with routers.
Message Brokers

Figure 4-21. The general organization of a message broker in a message-queuing system.

IBM’s WebSphere Message-Queuing System

Figure 4-22. General organization of IBM’s message-queuing system.

Channels

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport type</td>
<td>Determines the transport protocol to be used</td>
</tr>
<tr>
<td>FIFO delivery</td>
<td>Indicates that messages are to be delivered in the order they are sent</td>
</tr>
<tr>
<td>Message length</td>
<td>Maximum length of a single message</td>
</tr>
<tr>
<td>Setup retry</td>
<td>Specifies maximum number of retries to start up the remote MCA</td>
</tr>
<tr>
<td>Delivery retries</td>
<td>Maximum times MCA will try to put received message into queue</td>
</tr>
</tbody>
</table>

Figure 4-23. Some attributes associated with message channel agents.

Message Transfer (1)

Figure 4-24. The general organization of an MQ queuing network using routing tables and aliases.
Streams and Quality of Service

Properties for Quality of Service:
• The required bit rate at which data should be transported.
• The maximum delay until a session has been set up.
• The maximum end-to-end delay.
• The maximum delay variance, or jitter.
• The maximum round-trip delay.
Enforcing QoS (2)

Sent          1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Delivered     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Lost packet   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Gap of lost frames (b)

Figure 4-28. The effect of packet loss in (a) non-interleaved transmission and (b) interleaved transmission.

Synchronization Mechanisms (1)

Procedure that reads two audio data units for each video data unit

Incoming stream

Network

Application

Figure 4-29. The principle of explicit synchronization on the level data units.

Synchronization Mechanisms (2)

Application tells middleware what to do with incoming streams

Multimedia control is part of middleware

Middleware layer

Incoming stream

Receiver's machine

OS

Figure 4-30. The principle of synchronization as supported by high-level interfaces.

Overlay Construction

End host

Router

Overlay network

Internet

Figure 4-31. The relation between links in an overlay and actual network-level routes.
Information Dissemination Models (1)

- Anti-entropy propagation model
  - Node P picks another node Q at random
  - Subsequently exchanges updates with Q

- Approaches to exchanging updates
  - P only pushes its own updates to Q
  - P only pulls new updates from Q
  - P and Q send updates to each other

Information Dissemination Models (2)

Figure 4-32. The relation between the fraction s of update-ignorant nodes and the parameter k in pure gossipping. The graph displays ln(s) as a function of k.