Outline

- Middleware Communication Services
  - Remote Procedure Call
  - Message-Oriented Communication
    - Message-oriented transient communication
    - Message-oriented persistent communication
  - Stream-Oriented Communication
- Q&A
Conventional Procedure Call

(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 Stub: packs the parameters into a message and requests it to be sent
Server Stub: transforms the incoming requests into local procedure calls

Principle of RPC between a client and server program
Remote Procedure Calls (RPCs)

RPC Occurring 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.
RPCs (Cont’d)

- RPC Occurring in the Following Steps:

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

The steps involved in doing a remote computation through RPC

1. Client call to procedure
2. Stub builds message
3. Message is sent across the network
4. Server OS hands message to server stub
5. Stub unpacks message
6. Stub makes local call to "add"
Passing Value Parameters (Cont’d)

Illustration: Little Endian Format

- Numbering Bytes Right to Left

The original message on the Pentium

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved
Passing Value Parameters (Cont’d)

- **Illustration: Big Endian Format**
  - **Numbering Bytes Left to Right**

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

The message after receipt on the SPARC
Passing Value Parameters (Cont’d)

Illustration: Big Endian Format

The message after being inverted
The little numbers in boxes indicate the address of each byte

Need for additional information about what is a string and what is an integer!
Parameter Specification and Stub Generation

(a) A procedure

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

(b) The corresponding message

<table>
<thead>
<tr>
<th>foobar's local variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>z[0]</td>
</tr>
<tr>
<td>z[1]</td>
</tr>
<tr>
<td>z[2]</td>
</tr>
<tr>
<td>z[3]</td>
</tr>
<tr>
<td>z[4]</td>
</tr>
</tbody>
</table>

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved
Asynchronous RPC

The interaction between client and server in a traditional RPC
Asynchronous RPC (Cont’d)

The interaction using asynchronous RPC
A client and server interacting through two asynchronous RPCs
Example: DCE RPC

- **DCE (Distributed Computing Environment)**
  - Middleware System Designed to Execute as a Layer of Abstraction between Existing (Network) OSes and Distributed Applications
    - All communication between clients and servers takes place by means of RPC
  - One Providing Services
    - Distributed File Services
    - Directory Services
    - Security Services
    - Distributed Time Services
  - One Developed by Open Software Foundation Called Now Open Group
The steps in writing a client and a server in DCE RPC

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved
Writing a Client and a Server (Cont’d)

- Three Files Output by the IDL (Interface Definition Language) Compiler
  - A header file (e.g., interface.h, in C terms)
  - The client stub
  - The server stub
Binding a Client to a Server

- 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 (Cont'd)

Client-to-Server Binding in DCE
# Berkeley Sockets

<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>

The socket primitives for TCP/IP
Berkeley Sockets (Cont’d)

Connection-oriented communication pattern using sockets
Message-Passing Interface (MPI)

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_bsend</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_ssend</td>
<td>Send a message and wait until receipt starts</td>
</tr>
<tr>
<td>MPI_sendrecv</td>
<td>Send a message and wait for reply</td>
</tr>
<tr>
<td>MPI_isend</td>
<td>Pass reference to outgoing message, and continue</td>
</tr>
<tr>
<td>MPI_isend</td>
<td>Pass reference to outgoing message, and wait until receipt starts</td>
</tr>
<tr>
<td>MPI_recv</td>
<td>Receive a message; block if there is none</td>
</tr>
<tr>
<td>MPI_irecv</td>
<td>Check if there is an incoming message, but do not block</td>
</tr>
</tbody>
</table>

Some of the most intuitive message-passing primitives of MPI
Message-Queuing Model

Four combinations for loosely-coupled communications using queues

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved
General Architecture of a Message-Queueing System

The relationship between queue-level addressing and network-level addressing
General Architecture of a Message-Queuing System (Cont’d)

General organization of a message-queuing system with routers

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved
Message Brokers

The general organization of a message broker in a message-queuing system

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved
Data Stream

A general architecture for streaming stored multimedia data over a network
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

Using a buffer to reduce jitter
Enforcing QoS (Cont’d)

The effect of packet loss in (a) non interleaved transmission and (b) interleaved transmission

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved
Synchronization Mechanisms

The principle of explicit synchronization on the data units of simple streams

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved
The principle of synchronization as supported by high-level interfaces