Last Class: RPCs and RMI

• Case Study: Sun RPC

• Lightweight RPCs

• Remote Method Invocation (RMI)
  – Design issues

Today: Communication Issues

• Message-oriented communication
  – Persistence and synchronicity

• Stream-oriented communication
Persistence and Synchronicity in Communication

Persistence

• Persistent communication
  – Messages are stored until (next) receiver is ready
  – Examples: email, pony express
Transient Communication

- Transient communication
  - Message is stored only so long as sending/receiving application are executing
  - Discard message if it can’t be delivered to next server/receiver
  - Example: transport-level communication services offer transient communication
  - Example: Typical network router – discard message if it can’t be delivered next router or destination

Synchronicity

- Asynchronous communication
  - Sender continues immediately after it has submitted the message
  - Need a local buffer at the sending host

- Synchronous communication
  - Sender blocks until message is stored in a local buffer at the receiving host or actually delivered to sending
  - Variant: block until receiver processes the message

- Six combinations of persistence and synchronicity
Persistence and Synchronicity Combinations

a) Persistent asynchronous communication (e.g., email)
b) Persistent synchronous communication

c) Transient asynchronous communication (e.g., UDP)
d) Receipt-based transient synchronous communication
Persistence and Synchronicity Combinations

e) Delivery-based transient synchronous communication at message delivery (e.g., asynchronous RPC)

f) Response-based transient synchronous communication (RPC)

Message-oriented Transient Communication

- Many distributed systems built on top of simple message-oriented model
  - Example: Berkeley sockets
Berkeley Socket Primitives

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket</td>
<td>Create a new communication endpoint</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>

Message-Passing Interface (MPI)

- Sockets designed for network communication (e.g., TCP/IP)
  - Support simple send/receive primitives
- Abstraction not suitable for other protocols in clusters of workstations or massively parallel systems
  - Need an interface with more advanced primitives
- Large number of incompatible proprietary libraries and protocols
  - Need for a standard interface
- Message-passing interface (MPI)
  - Hardware independent
  - Designed for parallel applications (uses transient communication)
- Key idea: communication between groups of processes
  - Each endpoint is a (groupID, processID) pair
MPI Primitives

<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_issend</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 are none</td>
</tr>
<tr>
<td>MPI_irecv</td>
<td>Check if there is an incoming message, but do not block</td>
</tr>
</tbody>
</table>

Message-oriented Persistent Communication

- **Message queuing systems**
  - Support asynchronous persistent communication
  - Intermediate storage for message while sender/receiver are inactive
  - Example application: email
- **Communicate by inserting messages in queues**
- **Sender is only guaranteed that message will be eventually inserted in recipient’s queue**
  - No guarantees on when or if the message will be read
  - “Loosely coupled communication”
Message-Queuing Model

<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>

Stream Oriented Communication

- Message-oriented communication: request-response
  - When communication occurs and speed do not affect correctness
- Timing is crucial in certain forms of communication
  - Examples: audio and video (“continuous media”)
  - 30 frames/s video => receive and display a frame every 33ms
- Characteristics
  - Isochronous communication
    - Data transfers have a maximum bound on end-end delay and jitter
    - Push mode: no explicit requests for individual data units beyond the first “play” request
Examples

![Single sender and receiver](image)

![One sender, Multiple receivers](image)

Quality of Service (QoS)

- Time-dependent and other requirements are specified as quality of service (QoS)
  - Requirements/desired guarantees from the underlying systems
  - Application specifies workload and requests a certain service quality
  - Contract between the application and the system

<table>
<thead>
<tr>
<th>Characteristics of the Input</th>
<th>Service Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Maximum data unit size (bytes)</td>
<td>- Loss sensitivity (bytes)</td>
</tr>
<tr>
<td>- Token bucket rate (bytes/sec)</td>
<td>- Loss interval (µsec)</td>
</tr>
<tr>
<td>- Token bucket size (bytes)</td>
<td>- Burst loss sensitivity (data units)</td>
</tr>
<tr>
<td>- Maximum transmission rate (bytes/sec)</td>
<td>- Minimum delay noticed (µsec)</td>
</tr>
<tr>
<td>- Maximum bandwidth</td>
<td>- Maximum delay variation (µsec)</td>
</tr>
<tr>
<td>- Quality of guarantee</td>
<td></td>
</tr>
</tbody>
</table>
Specifying QoS: Token bucket

- The principle of a token bucket algorithm
  - Parameters (rate \( r \), burst \( b \))
  - Rate is the average rate, burst is the maximum number of packets that can arrive simultaneously

Setting Up a Stream: RSVP