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

- General organization of a communication system in which hosts are connected through a network.

Persistence

- Persistent communication
  - Messages are stored until (next) receiver is ready
  - Examples: email, pony express

Mail stored and sorted, to be sent out depending on destination and when pony and rider available.
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 (1)

<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>
General Architecture of a Message-Queuing System (2)

• Queue manager and relays
  – Relays use an overlay network
  – Relays know about the network topology and how to route

Message Brokers

• Message broker: application level gateway in MQS
  – Convert incoming messages so that they can be understood by destination (format conversion)
  – Also used for pub-sub systems
IBM’s WebSphere MQ

• Queue managers manage queues
  – Connected through message channels
• Message channel agent (MCA)
  – Checks queue, wraps into TCP packet, send to receiving MCA

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

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.
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></td>
<td>Maximum delay variation (μsec)</td>
</tr>
<tr>
<td></td>
<td>Quality of guarantee</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
Enforcing QoS

- Enforce at end-points (e.g., token bucket)
  - No network support needed
- Mark packets and use router support
  - Differentiated services: expedited & assured forwarding
- Use buffers at receiver to mask jitter
- Packet losses
  - Handle using forward error correction
  - Use interleaving to reduce impact

Enforcing QoS (1)
Enforcing QoS (2)

- Can also use forward error correction (FEC)

Stream synchronization

- Multiple streams:
  - Audio and video; layered video
- Need to sync prior to playback
  - Timestamp each stream and sync up data units prior to playback
- Sender or receiver?
- App does low-level sync
  - 30 fps: image every 33ms, lip-sync with audio
- Use middleware and specify playback rates
Synchronization Mechanism

Multicasting

- Group communication
  - IP multicast versus application-level multicast
  - Construct an overlay multicast tree rooted at the sender
  - Send packet down each link in the tree
- Issues: tree construction, dynamic joins and leaves
New Topic: Naming

- Names are used to share resources, uniquely identify entities and refer to locations
- Need to map from name to the entity it refers to
  - E.g., Browser access to www.cnn.com
  - Use name resolution
- Differences in naming in distributed and non-distributed systems
  - Distributed systems: naming systems is itself distributed
- How to name mobile entities?
Example: File Names

- Hierarchical directory structure (DAG)
  - Each file name is a unique path in the DAG
  - Resolution of /home/steen/mbox a traversal of the DAG
- File names are human-friendly

Resolving File Names across Machines

- Remote files are accessed using a node name, path name
- NFS mount protocol: map a remote node onto local DAG
  - Remote files are accessed using local names! (location independence)
  - OS maintains a mount table with the mappings
Name Space Distribution

- Naming in large distributed systems
  - System may be global in scope (e.g., Internet, WWW)
- Name space is organized hierarchically
  - Single root node (like naming files)
- Name space is distributed and has three logical layers
  - Global layer: highest level nodes (root and a few children)
    - Represent groups of organizations, rare changes
  - Administrative layer: nodes managed by a single organization
    - Typically one node per department, infrequent changes
  - Managerial layer: actual nodes
    - Frequent changes
  - Zone: part of the name space managed by a separate name server

Name Space Distribution Example

- An example partitioning of the DNS name space, including Internet-accessible files, into three layers.
Name Space Distribution

<table>
<thead>
<tr>
<th>Item</th>
<th>Global</th>
<th>Administrative</th>
<th>Managerial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geographical scale of network</td>
<td>Worldwide</td>
<td>Organization</td>
<td>Department</td>
</tr>
<tr>
<td>Total number of nodes</td>
<td>Few</td>
<td>Many</td>
<td>Vast numbers</td>
</tr>
<tr>
<td>Responsiveness to lookups</td>
<td>Seconds</td>
<td>Milliseconds</td>
<td>Immediate</td>
</tr>
<tr>
<td>Update propagation</td>
<td>Lazy</td>
<td>Immediate</td>
<td>Immediate</td>
</tr>
<tr>
<td>Number of replicas</td>
<td>Many</td>
<td>None or few</td>
<td>None</td>
</tr>
<tr>
<td>Is client-side caching applied?</td>
<td>Yes</td>
<td>Yes</td>
<td>Sometimes</td>
</tr>
</tbody>
</table>

- A comparison between name servers for implementing nodes from a large-scale name space partitioned into a global layer, as an administrational layer, and a managerial layer.
- The more stable a layer, the longer are the lookups valid (and can be cached longer)

Programming Assignment 1

- Goal: familiarity with RPCs, threads, sync, distributed application design
- Design a multi-tier micro-blogging site