Today
- Communication in Distributed Systems
  - Overview
  - Types
  - Remote Procedure Calls (RPC)
  - Remote Method Invocation (RMI)

Communication
- How do program modules/processes communicate on a single machine?

Communication in Distributed Systems
- "Distributed" processes
  - Located on different machines
  - Need communication mechanisms
  - Goal: Hide distributed nature as far as possible
Communication in Distributed Systems

- Networking primitives and protocols (e.g.: TCP/IP)
- Advanced communication models: Built on networking primitives
  - Remote Procedure Calls (RPC)
  - Remote Method Invocation (RMI)
  - Messages
  - Multicast

Types of Communication

- Defined by two main properties:
  - Persistence
  - Synchronization

Persistence

- Persistent communication
  - Messages are stored until receiver is ready
  - Sender/receiver don’t have to be up at the same time
- Transient communication
  - Message is stored only so long as both sending/receiving applications are executing
  - Discard message if it can’t be delivered to receiver

Synchronization

- Synchronous communication
  - Sender blocks until message is delivered to receiver
  - Variant: block until receiver processes the message
- Asynchronous communication
  - Sender continues immediately after it has submitted the message
- Several combinations of persistence and synchronization
**Persistence-Synchronization Combinations**

<table>
<thead>
<tr>
<th>Persistent Asynchronous communication</th>
<th>Persistent Synchronous communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>A sends message and continues</td>
<td>A stopped running</td>
</tr>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>B is not running</td>
<td>B starts and receives message (a)</td>
</tr>
<tr>
<td>Example: Email</td>
<td>Example: Message Queuing</td>
</tr>
</tbody>
</table>

**Persistence-Synchronization Combinations**

<table>
<thead>
<tr>
<th>Transient Asynchronous communication</th>
<th>Receipt-based Transient Synchronous Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>A sends message and continues</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A sends message</td>
</tr>
<tr>
<td>B</td>
<td>B receives message</td>
</tr>
<tr>
<td></td>
<td>B continues</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Example: UDP, One-way RPC</td>
<td>Example: Message-passing</td>
</tr>
</tbody>
</table>

**Remote Procedure Calls (RPC)**

- **Goal:** Make distributed computation look like centralized computation
- **Idea:** Allow processes to call procedures on other machines
  - Make it appear like normal procedure calls

**Persistence-Synchronization Combinations**

<table>
<thead>
<tr>
<th>Delivery-based Transient Synchronous Communication</th>
<th>Response-based Transient Synchronous Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send request and wait until accepted</td>
<td>Accepted</td>
</tr>
<tr>
<td>Request is received</td>
<td>Time</td>
</tr>
<tr>
<td>Running, but doing something else</td>
<td>Process request (e)</td>
</tr>
<tr>
<td>Example: Asynchronous RPC</td>
<td>Example: RPC, RMI</td>
</tr>
</tbody>
</table>
Local Procedure Calls

foo(i, buf)

Stack Pointer
buf
i
Return address, Local vars
Stack Pointer

RPC Operation

- Challenges:
  - Hide details of communication
  - Pass parameters transparently

- Stubs
  - Hide communication details
  - Client and server stubs
  - Marshalling
    - Flattening and parameter passing

Stubs

- Code that communicates with the remote side
- Client stub:
  - Converts function call to remote communication
  - Passes parameters to server machine
  - Receives results
- Server stub:
  - Receives parameters and request from client
  - Calls the desired server function
  - Returns results to client

Basic RPC Operation
Parameter Passing: Local Procedures

- Pass-by-value
  - Original variable is not modified
  - E.g.: integers, chars
- Pass-by-reference
  - Passing a pointer
  - Value may be changed
  - E.g.: Arrays
- Pass-by-copy/restore
  - Copy is modified and overwritten to the original
  - E.g.: in-out parameters in Ada

Parameter Passing: RPC

- Pass-by-value
  - Send the value in standard format
- Pass-by-reference
  - Can we pass pointers?
  - What about complex data structures (linked lists, trees, graphs)?

Marshalling

- Converting parameters into a byte stream
- Problems:
  - Heterogeneous data formats: Big-endian vs. little-endian
  - Type of parameter passing: by-value vs. by-reference

Heterogeneous Data Formats

- Use a standard data format
- Examples: Network byte order, XDR (Extended Data Representation)
- Decide on a protocol for parameter ordering
**Stub Generation**

- Most stubs are similar in functionality
  - Handle communication and marshalling
  - Differences are in the main server-client code
- Application needs to know only stub interface
- Interface Definition Language (IDL)
  - Allows interface specification
  - IDL compiler generates the stubs automatically

**Binding**

- How does the client stub find the server stub?
  - Needs to know remote IP address/port no.
- Port mapper
  - Daemon on server machine maintaining server bindings
  - Listens on a well-known port
- Server stub registers its port no. and service name with portmapper
  - Client gets this binding by querying portmapper
- Server’s IP address can be obtained from a directory service

**Synchronous RPC**

- RPC performed in a synchronous manner
  - Client blocks until results come back
- What if client wants to do something else?

**RPC Variants**

- Asynchronous RPC
  - Server sends ACK as soon as request is received
  - Executes procedure later
- Deferred synchronous RPC
  - Use two asynchronous RPCs
  - Server sends reply via second asynchronous RPC
  - Callback to the client
- One-way RPC
  - Client does not even wait for an ACK from the server
**Multicast RPC**

- RPC sent to multiple servers
  - Could use multiple concurrent one-way or asynchronous RPCs
- Why use multicast RPC?
- How to handle responses?
  - First one or majority response
  - Aggregate multiple responses

**Remote Method Invocation (RMI)**

- RPCs applied to distributed objects
- Class: object-oriented abstraction
- Object: instance of class
  - Encapsulates data
  - Exports methods: operations on data
  - Separation between interface and implementation

**Distributed Objects**

- Object resides on one machine, interfaces reside on other machines
- Remote object: State on different machine
- Local object: State on the same machine
- RMIs allow invoking methods of remote objects
  - Use proxies, skeletons, binding
  - Allow passing of object references as parameters

**Basic RMI Operation**

- Diagram showing the interaction between RMI Client and RMI Server with components like Proxy and Skeleton.
Proxies and Skeletons

- Proxy: client stub
  - Maintains server ID, endpoint, object ID
  - Does parameter marshalling
  - In practice, can be downloaded/constructed on the fly
- Skeleton: server stub
  - Does demarshalling and passes parameters to server
  - Sends result to proxy

Parameter Passing

- Less restrictive than RPCs
  - Supports system-wide object references
  - Copy local objects, pass references of remote objects

Binding a Client to an Object

- Loading a proxy in client address space
- Implicit binding:
  - Bound automatically on object reference resolution
- Explicit binding:
  - Client has to first bind object
  - Call method after binding