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

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
  - Streams

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

**Persistent Asynchronous communication**
- A sends message and continues
- B starts and receives message
Example: Email

**Persistent Synchronous communication**
- A sends message and waits until accepted
- B starts and receives message
Example: Message Queuing

**Transient Asynchronous communication**
- A sends message
- B receives message
Example: UDP, One-way RPC

**Transient Synchronous Communication**
- A sends message
- B receives message
Example: Message-passing

### 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
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

**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

**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)?
**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

**RPC Issues**
- RPC Performed in a synchronous manner
  - What if client wants to do something else?
- What if things fail?

**Asynchronous RPC**
- RPC: Client blocks until results come back
- 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
- One-way RPC
  - Client does not even wait for an ACK from the server
RPC Failure Semantics: Network failures

- Client unable to locate server:
  - Return error or raise exception
- Lost requests/replies:
  - Timeout mechanisms
  - Make operation idempotent
  - Use sequence numbers, mark retransmissions

RPC Failure Semantics: Server failure

- Server may crash during RPC
  - Did failure occur before or after operation?
- Operation semantics
  - Exactly once: desirable but impossible to achieve
  - At least once
  - At most once
  - No guarantee

RPC Failure Semantics: Client Failure

- Client crashes while server is computing
  - Server computation becomes orphan
- Possible actions
  - Extermination: log at client stub and explicitly kill orphans
  - Reincarnation: Divide time into epochs between failures and delete computations from old epochs
  - Expiration: give each RPC a fixed quantum T; explicitly request extensions