Outline

• Interprocess Communication in distributed systems
• Communication Models
  ➢ Remote Procedure Call (RPC)
  ➢ Remote Method Invocation (RMI)
  ➢ Message-Oriented Middleware
  ➢ Streams
**RPC: 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

\[ \text{count} = \text{read}(\text{fd}, \text{buf}, \text{nbytes}); \]

**Client and Server Stubs**

- Principle of RPC between a client and server program.
**Steps of a Remote Procedure Call**

1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS
9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client

**Passing Value Parameters**

- Steps involved in doing remote computation through RPC
- Passing reference parameters?
Parameter Specification and Stub Generation

- Both sides follow the same protocol
  - Message format (parameter marshalling)
  - Representation of simple data structures
  - Actual exchange of messages
- Implementation of client and server stubs
- Interfaces specified by an Interface Definition Language (IDL)
- Interface compiled into a client stub and a server stub (with compile-time and run-time interfaces)
  - Through an IDL compiler

Asynchronous RPC 1/2

a) The interconnection between client and server in a traditional RPC
b) The interaction using asynchronous RPC
Asynchronous RPC 2/2

- A client and server interacting through two asynchronous RPCs

Distributed Objects

- Common organization of a remote object with client-side proxy.
Binding a Client to an Object

(a) Example with implicit binding using only global references

Distr_object* obj_ref; // Declare a systemwide object reference
obj_ref = ...; // Initialize the reference to a distributed object
obj_ref-> do_something(); // Implicitly bind and invoke a method

(b) Example with explicit binding using global and local references

Distr_object objRef; // Declare a systemwide object reference
Local_object* obj_ptr; // Declare a pointer to local objects
obj_ref = ...; // Initialize the reference to a distributed object
obj_ptr = bind(obj_ref); // Explicitly bind and obtain a pointer to the local proxy
obj_ptr -> do_something(); // Invoke a method on the local proxy

Static versus Dynamic RMI

- Client bound to an object
  - Invoke object’s methods through proxy (RMI)
- RMI supports system-wide object references
- Object-specific stubs
- Object interface provided through IDL
- Use an object-based language (such as Java) to handle stub generation automatically
  - Static invocation
    - use predefined interface definitions
    - Object’s interface known when client application developed
    - If interface changes, client application recompiled
Static versus Dynamic RMI 2/2

- **Dynamic invocation**
  - Compose a method invocation at runtime
  - Application selects at runtime which method it will invoke at a remote object
  - `invoke(object, method, input_parameters, output_parameters)`
- Static: `fobject.append(int)`
- Dynamic: `invoke(fobject, id(append), int)`

RMI: Parameter Passing

- The situation when passing an object by reference or by value.
Message-Oriented Communication

- Inherent synchronous nature of RPC and RMI
- Messaging
  - Synchronous
    - Sender blocked until message stored in a local buffer at receiving host or actually delivered to receiver (could block until message processed)
  - Asynchronous
    - Sender continues immediately after message submitted for transmission
    - Message stored in communication system
  - Persistent (email)
    - Message stored by communication system as long as it takes to deliver to receiver
  - Transient (store-and-forward routers, sockets)
    - Message stored by communication system only as long as the sending and receiving application are executed

Persistence and Synchronicity in Communication 1/4

General organization of a communication system in which hosts are connected through a network
a) Persistent asynchronous communication
b) Persistent synchronous communication
c) Transient asynchronous communication
d) Receipt-based transient synchronous communication
Persistence and Synchronicity in Communication 4/4

e) Delivery-based transient synchronous communication at message delivery

f) Response-based transient synchronous communication

General Architecture of a Message-Queuing System 1/2

- The relationship between queue-level addressing and network-level addressing.
General Architecture of a Message-Queuing System 2/2

- The general organization of a message-queuing system with routers.

Message Brokers

The general organization of a message broker in a message-queuing system.
Stream-Oriented Communication

• Timing is a problem
• Support for continuous media (temporal relationships between different data items are important to correctly interpret what the data means)
• Data streams transmission modes
  ➢ Asynchronous
  ➢ Synchronous (maximum end-to-end delay for each unit in a data stream)
  ➢ Isochronous (on-time, maximum and minimum end-to-end delay, bounded jitter)
• Need for QoS