Summary of Interprocess Communication and Coordination

Kelly Westbrooks
Department of Computer Science
Georgia State University

February 22, 2007
Outline

- Outline
- Message Passing Communication
- Request/Reply Communication
- Transaction Communication
- Name and Directory Services
- Distributed Mutual Exclusion
Basic Communication Primitives

The primitive functions for message passing-based communication are:

- \texttt{send(destination, message)}
- \texttt{receive(source, message)}

Both the \textit{source} and \textit{destination} can be addressed in 4 different ways:

- process name
- link
- mailbox
- port
The sending and receiving of messages can occur in two different ways:

- blocking
- nonblocking

Delivery of the message also can be sorted into two categories:

- reliable
- unreliable
Pipe and Socket APIs

pipe  Used for interprocess communication within a single machine.

socket  Used for interprocess communication between two different machines.
Secure Sockets

Secure sockets use public-key cryptography to encrypt the message payload. They can be used with a variety of application protocols (examples: https, sftp).
Multicast is used for one-to-many communication on a network. It can be either reliable, unreliable, or *best-effort*. Best-effort multicast ensures that at least one machine receives the message broadcasted. Messages have three different orderings:

**FIFO ordering**  Multicast messages from a single source are delivered in the order that they were sent.

**causal ordering**  Causally-related messages from multiple sources are delivered in their causal order.

**total ordering**  All messages from multiple sources are delivered in the same order they are sent.
RPC Operations

RPC = Remote Procedure Call An implementation of RPC addresses the following issues:

- Parameter-passing and data conversion
- Binding
- Compilation
- Exception and failure handling
- Security
Parameter-passing and data conversion

The method of parameter-passing can be one of four types:

- Call-by-value
- Call-by-name
- Call-by-reference
- Call-by-copy/restore
Binding

1. Server registers itself with the *port-mapper*
2. Client contacts the port-mapper to obtain a *handle* to the remote process
3. Port-mapper returns the port number of the remote process to the client
4. Client system builds the *client handle*. This handle is used to contact the remote process
Compilation

1. Programmer specifies the interface in using an IDL (interface definition language)
2. IDL is compiled into three things:
   - Header file
   - Server stub
   - Client stub
3. The header, server stub, and the server source code are compiled into the server.
4. The header, client stub, and client source code are compiled into the client.
5. Both the server and client use an RPC runtime component to facilitate RPC during execution.
RPC Exception and Failure Handling

Fundamental questions:

- How does the server report status information to the clients?
- How does a client send control information to the server?

RPC call semantics in the presence of failures:

- Server raises exception and the client retries when the server recovers
- Server raises exception and the client gives up
- Server does not report an error and the client retries the request
Orphaned computation can be eliminated in the following ways:

- By the client: upon restart, the client cleans up any orphaned computations.
- By the server: periodically, the server tries to locate owners of computations and discards the computation if the owner cannot be found.
- By expiration: Every computation has a timeout value associated with it.
Secure RPC

Two reasons for the need for Secure RPC:

1. RPC introduces vulnerabilities because it explicitly allows for the execution of code by remote machines.
2. RPC is fundamental to client/server communication.

Any RPC protocol needs to provide the following features:

- Mutual authentication
- Message integrity
- Message confidentiality
- Message originality
The ACID Properties

**Atomicity**
Either all operations in a transaction are performed or none are performed, even in the presence of failures.

**Consistency**
The execution of interleaved transactions is equivalent to the execution of the same transactions in serial order.

**Isolation**
Partial results of an incomplete transaction are not visible to others until the transaction is complete.

**Durability**
The system guarantees that once a transaction has been committed, its results are permanent even if a failure occurs after commitment.
The Two-phase Commit Protocol

1. Phase 1
   1.1 Coordinator precommits the transaction
   1.2 Coordinator sends the transaction request to all participants
   1.3 Participant receives request message
   1.4 If participant agrees with the transaction, it sends YES back to coordinator. Otherwise, it sends NO.
   1.5 Coordinator collects all replies from Participants.

2. Phase 2
   2.1 If all votes are unanimous YES, the commit the transaction and send COMMIT to all participants. Otherwise abort and send ABORT.
   2.2 Participant receives decision from Coordinator
   2.3 If COMMIT, then participant commits transaction and send FINISH. Otherwise, participant aborts transaction and send FINISH.
   2.4 Coordinator waits to receive all FINISH messages from participants.
Name and Address Resolution

The resolution process has two steps:

1. **Name resolution** - turns a name into an address
2. **Address resolution** - turns an address into a network route
Object Attributes and Name Structures

Two types of resolution:

**heirarchical structure name-based resolution** resolved by traversing the namespace in a treelike fashion.

**structure-free attribute-based resolution** resolved by querying the space based upon attribute values.
The DIT (Directory Information Tree) can be decomposed into two classes:

**naming domain** name subspace for which there is a single administrative authority.

**naming context** a partial subtree of the DIT

There are two participants in name resolution:

- **DUA** Directory User Agent
- **DSA** Directory Service Agent
Contention-based Mutual Exclusion

How to break ties when multiple requests for the same resource occur simultaneously?

- Timestamp Prioritized
- Voting
Token-based Mutual Exclusion

Token-based approaches use a token to determine who has access to the shared resource. There are three possible topologies:

**Ring structure** When the process with the token is finished with the critical section, it passes the token to the next process in the ring.

**Tree structure** Token is always at the root of the tree. A process requests the token by messaging the root. If successful, that processes node becomes the new root of the tree.

**Broadcast structure** A process broadcasts the token request to all other processes. When the token becomes available, it it transfered to the requestor.