CHAPTER 4: INTERPROCESS COMMUNICATION AND COORDINATION

Chapter outline

• Discuss three levels of communication: basic message passing, request/reply and transaction communication based on message passing
• Discuss name services for communication
• Show examples of process coordination using message passing

Basic message passing communication

Communication primitives:

send(destination, message)
receive(source, message)
channel naming = process name, link, mailbox, port

• direct communication: symmetric/asymmetric process naming, link
• indirect communication: many-to-many mailbox, many-to-one port
Message buffering and synchronization

1. **Nonblocking send, 1+8**: Sender process is released after message has been composed and copied into sender’s kernel (local system call).

2. **Blocking send, 1+2+7+8**: Sender process is released after message has been transmitted to the network (NIC interrupt OS).

3. **Reliable blocking send, 1+2+3+6+7+8**: Sender process is released after message has been received by the receiver’s kernel (kernel receives network ACK).

4. **Explicit blocking send, 1+2+3+4+5+6+7+8**: Sender process is released after message has been received by the receiver process (kernel receives kernel delivery ACK).

5. **Request and reply, 1-4, service, 5-8**: Sender process is released after message has been processed by the receiver and response returned to the sender.
Message passing API

- *Pipe*: A FIFO byte-stream unidirectional link for related processes (set up at process invocation)
- *Message queue*: A structured variable length message queue
- *Named Pipe*: A special FIFO file pipe using path name for unrelated processes under the same domain (explicitly created and accessed)
- *Socket*: A logical communication endpoint for communication between autonomous domains (bound to physical communication endpoint)
Connectionless socket communication

- **peer process**: application level processes - application protocol
- **LCE**: Logical Communication Endpoint - established with socket call
- **PCE**: Physical Communication Endpoint - (Transport TSAP/L4SAP, Network NSAP/L3SAP) bound to LCE with bind call
- **Network**: Accessed by sendto/recvfrom primitives
Connection-oriented socket communication
Asymmetric - Client and Server

Server starts first:

- **Server process**: application level process - server protocol
- **LCE**: Logical Communication Endpoint - established with socket call
- **PCE**: Physical Communication Endpoint - (Transport TSAP/L4SAP, Network NSAP/L3SAP) bound to LCE with bind call
- **Listen**: Server waits for incoming connection request
- **Accept**: Server accepts connection request, initializes connection
- **Read**: Server reads incoming segment(s) of request
- **Write**: Server writes reply segment(s)
- **Close**: Server terminates connection when reply is received

Client starts after Server:

- **Client process**: application level process - runs server protocol
- **LCE**: Logical Communication Endpoint - established with socket call
- **PCE**: Physical Communication Endpoint - (Transport TSAP/L4SAP, Network NSAP/L3SAP) bound to LCE with connect call, which also initialized connection to server PCE
- **Write**: Client writes request segment(s)
- **Read**: Client reads incoming segment(s) of reply
- **Close**: Client terminates connection when reply is received and acknowledged
Secure Socket Layer protocol

- **Privacy**: use symmetric private-key cryptography
- **Integrity**: use message integrity check
- **Authenticity**: use asymmetric public-key cryptography

### CLIENT

- **ClientHello**: randomC, CipherSuites
- **ClientKeyExchange**: randomS, CipherSuite, session id, server public key
- **ChangeCipherSpec**
- **Finished**: hashed message and secret
- **Socket Message**: encrypted and signed

### SERVER

- **ServerHello**: ServerKeyExchange
- **Finished**
- **Socket Message**

- Server accepts connection, selects a cipher suite both can use (if any), provides its public key in a signed certificate
- Client verifies server public key certificate
- Client and Server exchange public information to establish shared secret
- Client and Server initialize hash key, session encryption key
- Either Client or Server may terminate secure connection
Group communication and multicast

- Reliability of message delivery
  - Best effort
  - Duplicate detection
  - Omission detection/recovery per receiver
  - All or none (atomic) to all receivers

- Orderly delivery
  - FIFO (per sender)
  - Causal order
  - Total order
• (a) Single sender/single group - reliable, ordered delivery (FIFO)

• (b) Multiple senders/single group - order between senders’ messages?

• (c-L) Single sender/overlapping groups - order between messages sent to different groups for nodes in intersection - consistent?

• (c-R) Multiple, single group senders/overlapping groups - order between messages for nodes in intersection - consistent?

• (d-L) Multiple, multi-group senders/independent groups - issues of (b), plus consistency of order in Group 1 and in Group 2

• (d-R) Multiple, multi-group senders/overlapping groups - issues of (d-L), plus consistency of order for nodes in intersection of Group 1 and Group 2
Causal order

- Accept message \( m \) if \( T_i = S_i + 1 \) and \( T_k \leq S_k \) for all \( k \neq i \).
- Delay message \( m \) if \( T_i > S_i + 1 \) or there exists a \( k \neq i \) such that \( T_k > S_k \).
- Reject the message if \( T_i \leq S_i \).

<table>
<thead>
<tr>
<th>Message</th>
<th>Tx Timestamp</th>
<th>Rx Event</th>
<th>Action(s)</th>
<th>VLC after Rx</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1_1</td>
<td>000</td>
<td>R1_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M1_2</td>
<td>000</td>
<td>R1_2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M2_1</td>
<td>000</td>
<td>R1_3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M3_1</td>
<td>000</td>
<td>R1_4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M3_2</td>
<td>000</td>
<td>R2_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M3_3</td>
<td>000</td>
<td>R2_2</td>
<td>R2_3</td>
<td>R2_4</td>
</tr>
<tr>
<td></td>
<td>000</td>
<td>R3_1</td>
<td>R3_2</td>
<td>R3_3</td>
</tr>
</tbody>
</table>
Total order

Total Order Multicast Example: Time–Space Diagram
Request/reply communication

Remote Procedure Calls (RPCs)

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

1. **register service**

2. **create**

3. **port #**

4. **client handle**

**client** → **directory server** (binder or trader) → **port mapper** → **server**

**server machine**

**client machine**

**server machine address** or handle to server

**register program, version, and port**
RPC compilation

server procedures

RPC generator

interface specification

client main program

server stub

header file

client stub

RPC run-time library

compilation

server program

compilation

client program
RPC exception and failure

- **Exception**: in-band or out-band signaling

- **Link failure**: retransmission, sequence number and idempotent requests, use of transaction id \( xid \)

- **Server crash**:
  - *at least once*: server raises an exception and client retries
  - *at most once*: server raises an exception and client gives up
  - *maybe*: server raises no exception and client retries

- **Client crash**:
  - orphan killed by client
  - orphan killed by server
  - orphan killed by expiration
Secure RPC

- $C_s$ and $S_s$ are 128-bit random numbers.
- $C_p = \alpha^{C_s} \mod M$, and $S_p = \alpha^{S_s} \mod M$, where $\alpha$ and $M$ are known constants.

$$
SK_{cs} = S_p^{C_s} = (\alpha^{S_s})^{C_s} = \alpha^{S_s \cdot C_s}
$$

$$
SK_{sc} = C_p^{S_s} = (\alpha^{C_s})^{S_s} = \alpha^{C_s \cdot S_s}
$$
Transaction Communication

ACID properties

- Atomicity
- Consistency
- Isolation
- Durability

Two-phase commit protocol

**COORDINATOR**
- precommit the transaction
- send request to all participants
- collect all replies
- if all votes are unanimous YES then commit and send COMMIT else abort and send ABORT
- received response

**PARTICIPANT**
- received request message
- if ready then precommit and send YES else abort transaction and send NO
- reply
- decision
- receive decision
- if COMMIT then commit
- if ABORT then abort
- result
- send response
Failure and recovery of the 2PC protocol

Coordinator failure recovery actions

<table>
<thead>
<tr>
<th>begin</th>
<th>pre-commit</th>
<th>send request</th>
<th>collect</th>
<th>commit</th>
<th>send commit</th>
<th>receive</th>
<th>commit response</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>abort</td>
<td>abort or continue</td>
<td>resend commit message</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Participant failure recovery actions

<table>
<thead>
<tr>
<th>begin</th>
<th>receive request</th>
<th>commit</th>
<th>reply</th>
<th>send commit</th>
<th>receive commit</th>
<th>send resp.</th>
<th>update</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>abort</td>
<td>find out commit or abort</td>
<td>continue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Name and Directory Services

Object attributes and name structures

<table>
<thead>
<tr>
<th>Service /object Attributes</th>
<th>Name Structures</th>
<th>Attribute Partitioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; attributes &gt;</td>
<td>flat structure</td>
<td>physical</td>
</tr>
<tr>
<td>&lt; name, attributes, address &gt;</td>
<td>hierarchical structure name-based resolution (e.g., white pages)</td>
<td>organizational</td>
</tr>
<tr>
<td>&lt; name, type, attributes, address &gt;</td>
<td>structure-free attribute-based resolution (e.g., yellow pages)</td>
<td>functional</td>
</tr>
</tbody>
</table>

Name space and information base

Five naming contexts of Directory Info Tree in three Directory Service Agents
Name resolution

Recursive chaining

Transitive chaining

Referral

Multicast
Distributed Mutual Exclusion

- *Contention-based:*
  - Timestamp prioritized
  - Voting
- *Control (Token)-based:*
  - Ring structure
  - Tree structure
  - Broadcast structure

Tree-structure token passing
Broadcast structure token passing

<table>
<thead>
<tr>
<th>Process 1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15</td>
<td>20</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>Process 2</td>
<td>14</td>
<td>21</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>Process 3</td>
<td>15</td>
<td>21</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>Process 4</td>
<td>15</td>
<td>21</td>
<td>10</td>
<td>9</td>
</tr>
</tbody>
</table>

Sequence vectors $S_i$

<table>
<thead>
<tr>
<th>Process 1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15</td>
<td>20</td>
<td>10</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>Process 2</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 3</td>
<td>15</td>
<td>20</td>
<td>11</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>Process 4</td>
<td>15</td>
<td>20</td>
<td>11</td>
<td>9</td>
<td>2</td>
</tr>
</tbody>
</table>

Token vector $T$

Token queue $Q$
Leader Election

Complete topology
The Bully algorithm

- Are-U-Up to higher numbered nodes
- If highest alive, Enter-Election to lower nodes
- When ACK or TRO for all lower nodes, send Result
- Enter-Election received: transient state until Result

Logic ring topology
The initiator node sets participating = true and send (id) to its successor node;

For each process node,

receive (value);

case
    value > id : participating := true, send (value);
    value < id and participating == false : participating := true, send (id);
    value == id : announce leader;
end case

Tree topology
Distributed MST formation, timestamp protocol