Interprocess Communication

- Primitives
- Message Passing: issues
- Communication Schemes

Reading: Colouris, Chapter 4

---

Interprocess Communication (IPC)

lack of shared memory => communicate by sending messages

Primitives for interprocess communication
- message passing
  - the RISC among the IPC primitives
- remote procedure call (RPC)
  - process interaction at language level
  - type checking
- transactions
  - support for operations and their synchronization on shared objects
Message Passing

- The primitives:
  
  ```
  send expression_list to destination_identifier;
  receive variable_list from source_identifier;
  ```

- Variations:
  
  ```
  guarded receive:
  receive variable_list from source_id when B;
  
  selective receive:
  select
  receive var_list from source_id1;
  receive var_list from source_id2;
  receive var_list from source_id3;
  end
  ```

Semantics of Message-Passing Primitives

- blocking vs. non-blocking
- buffered vs. unbuffered
- reliable vs. unreliable
- fixed-size vs. variable-size messages
- direct vs. indirect communication
## Blocking vs. Non-Blocking Primitives

<table>
<thead>
<tr>
<th></th>
<th>blocking</th>
<th>non-blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>send</td>
<td>Returns control to user only after message has been sent, or until acknowledgment has been received.</td>
<td>Returns control as soon as message queued or copied.</td>
</tr>
<tr>
<td>receive</td>
<td>Returns only after message has been received.</td>
<td>Signals willingness to receive message. Buffer is ready.</td>
</tr>
<tr>
<td>problems</td>
<td>• Reduces concurrency.</td>
<td>• Need buffering:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• still blocking</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• deadlocks!</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Tricky to program.</td>
</tr>
</tbody>
</table>

## Buffered vs. Unbuffered Primitives

- Asynchronous `send` is never delayed
  - may get arbitrarily ahead of `receive`.
- However: messages need to be buffered.
- If no buffering available, operations become blocking, and processes are synchronized on operations: **rendezvous**.
**Reliable vs. Unreliable Primitives**

- Transmission problems: corruption, loss, duplication, reordering
- Recovery mechanism: Where?
- Reliable transmission: acknowledgments

![Diagram showing reliable transmission](image)

- At-least-one vs. exactly-one semantics

![Diagram showing at-least-one vs. exactly-one semantics](image)

**Direct vs. Indirect Communication**

- Direct communication:

  ```
  send(P, message)
  receive(Q, message)
  ```

- Variation thereof:

  ```
  send(P, message)
  receive(var, message)
  ```

![Diagram showing direct vs. indirect communication](image)
Indirect communication:
- Treat communication paths as first-class objects.

**Mailboxes:**

\[
\begin{align*}
&\text{send}(M, \text{msg}_1) \quad \text{send}(M, \text{msg}_1) \quad \text{send}(M, \text{msg}_1) \\
&\text{receive}(M, \&\text{msg}) \quad \text{receive}(M, \&\text{msg}) \quad \ldots
\end{align*}
\]

Indirect communication (cont)

**Ports:**
- example: Accent (CMU)

- multiple senders
- only one receiver
- access to port is passed between processes in form of capabilities
Communication Schemes

- one-to-one unicast
- one-to-many multicast
- many-to-one
- many-to-many