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
### Interprocess Communication

#### Blocking vs. Non-Blocking Primitives

<table>
<thead>
<tr>
<th></th>
<th>blocking</th>
<th>non-blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>send</strong></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><strong>receive</strong></td>
<td>Returns only after message has been received.</td>
<td>Signals willingness to receive message. Buffer is ready.</td>
</tr>
</tbody>
</table>
| **problems**  | - *Reduces concurrency.*                                                  | - *Need buffering:*  
|               |                                                                           |     *still blocking*  
|               |                                                                           |     *deadlocks!*            
|               |                                                                           |     *Tricky to program.*     |

#### 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**.

---

**invoke** entry  
**copy input parms**  
**accept** on entry  
**rendezvous**  
**copy output parms**

**invoke** entry  
**copy input parms**  
**accept** on entry  
**rendezvous**  
**copy output parms**
### Reliable vs. Unreliable Primitives

- **Transmission problems:**
  - corruption
  - loss
  - duplication
  - reordering

- **Recovery mechanism:** Where?
- **Reliable transmission:** acknowledgments

![Diagram of reliable vs. unreliable transmission]

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

  ![Diagram of at-least-one vs. exactly-one semantics]

### Direct vs. Indirect Communication

- **Direct communication:**
  - `send(P, message)`
  - `receive(Q, message)`

- **Variation thereof:**
  - `send(P, message)`
  - `receive(var, message)`

![Diagram of direct vs. indirect communication]

- `send(S, msg)`
  - `receive(&client_id, &msg)`
  - `server`

- `C_1`

- `C_2`

- `send(S, msg)`
  - `receive(&client_id, &msg)`

- `request table`
Indirect Communication

- Treat communication paths as first-class objects.

- Example: Mailboxes

  \[\text{send}(M, \text{msg}_1)\]
  \[\text{send}(M, \text{msg}_2)\]
  \[\text{send}(M, \text{msg}_3)\]

  \[\text{receive}(M, \&\text{msg})\]
  \[\text{receive}(M, \&\text{msg})\]

  mailbox \( M \)

  \[\ldots\]

Indirect Communication (2)

- Example: Accent (CMU)

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

  \[\text{send}(P, \text{msg}_1)\]
  \[\text{send}(P, \text{msg}_2)\]

  \[\text{receive}(P, \&\text{msg})\]
Communication Schemes

- One-to-one unicast
- One-to-many multicast
- Many-to-one
- Many-to-many