Message Passing

Channels

Message Passing

- Communication links can be established between threads/processes. There are three important issues:
- **Naming**: How to refer to each other?
- **Synchronization**: Shall we wait when participating a message activity?
- **Buffering**: Can message wait in a communication link?

Naming: Direct Communication Symmetric Scheme

- **Direct Communication: Each process that wants to communicate must explicitly name the other party:**
  - `Send(receiver, message);`
  - `Receive(sender, message);`
- **With this scheme:**
  - **A link is established between exactly two processes.**
  - **Exactly one link exists between each pair of processes.**
  - **We may establish these links for those processes that want to communicate before they run**

Naming: Direct Communication
Asymmetric Scheme

- **In this scheme, we have**
  - `Send(receiver, message);`
  - `Receive(id, message);`
- **The `Receive()` primitive receives the ID of the sender. Thus, in this scheme, a receiver can receive message from any process.**

Naming: Direct Communication

- There are disadvantages in this symmetric and asymmetric schemes:
  - Changing the name/ID of a process may require examining all other process definitions.
  - Processes must know the IDs of the other parties to start a communication.

Naming: Indirect Communication

- With indirect communication, the messages are sent to and received from *mailboxes* or *ports*.
- Each mailbox has a unique ID.
- The primitives are
  - `Send(mailbox-name, message);`
  - `Receive(mailbox-name,message);`
- Thus, messages are sent to and received from mailboxes.

Naming: Indirect Communication
Communication Links

- There is a link between two processes only if they share a mailbox.
- A link may be shared by multiple processes.
- Multiple links may exist between each pair of processes, with each link corresponding to a mailbox.

Naming: Indirect Communication
Communication Links

- What if there is only one message in a mailbox and two processes execute `Receive()`? It depends on the following:
  - Only one link between at most two processes
  - Allow at most one process to receive at a time
  - Allow the system to select an arbitrary order

Synchronization

- The sender and receiver may be blocked:
  - **Blocking Send**: the sender blocks until its message is received
  - **Nonblocking Send**: the sender sends and resumes its execution
  - **Blocking Receive**: the receiver blocks until a message is available
  - **Nonblocking Receive**: the receive receives a message or a null.
- When both send and receive are blocking, we have a *rendezvous* between the sender and receiver.

Synchronous vs. Asynchronous

- ***Blocking*** **and** ***non-blocking*** **are known as synchronous and asynchronous.**
  - **If the sender and receiver must synchronize their activities, use synchronous communication.**
  - **Because the uncertainty in the order of events, asynchronous communication is more difficult to program.**
  - **On the other hand, asynchronous algorithms are general and portable, because they are guaranteed to run correctly in networks with arbitrary timing behavior.**

Capacity

- **The *capacity* of a communication link is its buffer size:**
  - **Zero Capacity: Since no message can be waiting in the link, it is synchronous. Sender may block**
  - **Unbounded Capacity: Messages can wait in the link. Sender never blocks and the link is asynchronous. *The order of messages being received does not have to be FIFO*.**
  - **Bounded Capacity: Buffered Message Passing. Sender blocks if the buffer is full, and the link is asynchronous.**