Inter-process Communication
Interprocess Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - `send(message)` – message size fixed or variable
  - `receive(message)`
- If P and Q wish to communicate, they need to:
  - establish a communication link between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)
Implementation Questions

- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?
- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?
Communications Models

Message Passing  Shared Memory

(a)  (b)

process A  process A

process B  shared

kernel  process B

kernel
Direct Communication

- Processes must name each other explicitly:
  - \texttt{send} \((P, \text{message})\) – send a message to process \(P\)
  - \texttt{receive} \((Q, \text{message})\) – receive a message from process \(Q\)

- Properties of communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional, but is usually bidirectional
Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports)
  - Each mailbox has a unique id
  - Processes can communicate only if they share a mailbox

- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional
Indirect Communication

- Operations
  - create a new mailbox
  - send and receive messages through mailbox
  - destroy a mailbox

- Primitives are defined as:

  $\text{send}(A, message) \rightarrow$ send a message to mailbox $A$
  
  $\text{receive}(A, message) \rightarrow$ receive a message from mailbox $A$
Indirect Communication

- Mailbox sharing
  - $P_1$, $P_2$, and $P_3$ share mailbox A
  - $P_1$, sends; $P_2$ and $P_3$ receive
  - Who gets the message?

- Solutions
  - Allow a link to be associated with at most two processes
  - Allow only one process at a time to execute a receive operation
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.
  - Each receiver receives a copy of the message (publish/subscribe)
Synchronization

- Message passing may be either blocking or non-blocking
- **Blocking** is considered **synchronous**
  - **Blocking send** has the sender block until the message is received
  - **Blocking receive** has the receiver block until a message is available
- **Non-blocking** is considered **asynchronous**
  - **Non-blocking send** has the sender send the message and continue
  - **Non-blocking receive** has the receiver receive a valid message or null
Buffering

- Queue of messages attached to the link; implemented in one of three ways
  1. Zero capacity – 0 messages
     Sender must wait for receiver (rendezvous)
  2. Bounded capacity – finite length of $n$ messages
     Sender must wait if link full
  3. Unbounded capacity – infinite length
     Sender never waits
Client-Server Communication

- Sockets
- Remote Procedure Calls
- Remote Method Invocation (Java)
Sockets

- A socket is defined as an endpoint for communication
- Concatenation of IP address and port
- The socket 161.25.19.8:1625 refers to port 1625 on host 161.25.19.8
- Communication consists of a pair of sockets
Socket Communication

host X
(146.86.5.20)

socket
(146.86.5.20:1625)

web server
(161.25.19.8)

socket
(161.25.19.8:80)
Remote Procedure Calls

- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems.
- **Stubs** – client-side proxy for the actual procedure on the server.
- The client-side stub locates the server and *marshalls* the parameters.
- The server-side stub receives this message, unpacks the marshalled parameters, and performs the procedure on the server.
- It then marshalls the response and sends it back to the client-side stub, which unpacks the results and returns them to the client.
Execution of RPC

```
user calls kernel to send RPC message to procedure X

kernel sends message to matchmaker to find port number

matchmaker receives message, looks up answer

matchmaker replies to client with port P

daemon listening to port P receives message

daemon processes request and processes send output

From: client
To: server
Port: matchmaker
Re: address for RPC X

From: server
To: client
Port: kernel
Re: RPC X
Port: P

from: client
To: server
Port: port P
<contents>

kernel places port P in user RPC message

kernel sends RPC

kernel receives reply, passes it to user

daemon processes request and processes send output
```
Remote Method Invocation

- Remote Method Invocation (RMI) is a Java mechanism similar to RPCs.
- RMI allows a Java program on one machine to invoke a method on a remote object.
Marshalling Parameters

client

\[\text{val} = \text{server.someMethod}(A, B)\]

remote object

\[
\text{boolean someMethod (Object } x, \text{ Object } y) \\
\{ \quad \text{implementation of someMethod} \\
\quad \ldots \\
\}
\]

stub

skeleton

A, B, someMethod

boolean return value