Cooperating Processes

- Independent process cannot affect or be affected by the execution of another process
- Cooperating process can affect or be affected by the execution of another process
- Why cooperating?
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

Example: Producer-Consumer Problem

- Paradigm for cooperating processes, producer process produces information that is consumed by a consumer process

![Diagram of Producer-Consumer Problem]
Inter-Process Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions
- Shared Memory Systems
  - A region of memory are shared among processes
- Message-Passing Systems
  - Message exchange for communication
- Comparison between shared memory and message-passing
  - Shared memory is more efficient
  - Message-passing is easy for programming
  - More?

Communication Models

(a) process A
   M

(b) process A
   shared
   process B
   M

(a) process A
   kernel
   M

(b) process A
   kernel

Oct. 01, 2007
CSE660: Introduction to Operating Systems
Message-Passing Systems

- 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 be there 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?
Direct Communication

- Processes must name each other explicitly:
  - `send (P, message)` – send a message to process P
  - `receive (Q, 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 bi-directional

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:
  - send\((A, message)\) – send a message to mailbox A
  - receive\((A, message)\) – receive a message from mailbox A

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
  - Zero capacity – 0 messages
    - Sender must wait for receiver (rendezvous)
  - Bounded capacity – finite length of $n$ messages
    - Sender must wait if link full
  - Unbounded capacity – infinite length
    - Sender never waits

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 between a pair of sockets
Socket Communication

Remote Procedure Calls

- Why?
- 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.
Remote Method Invocation

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

```java
val = server.someMethod(A,B)

boolean someMethod (Object x, Object y)
{
    implementation of someMethod
    ...
}
```

- stub
- skeleton
- A, B, someMethod
- boolean return value