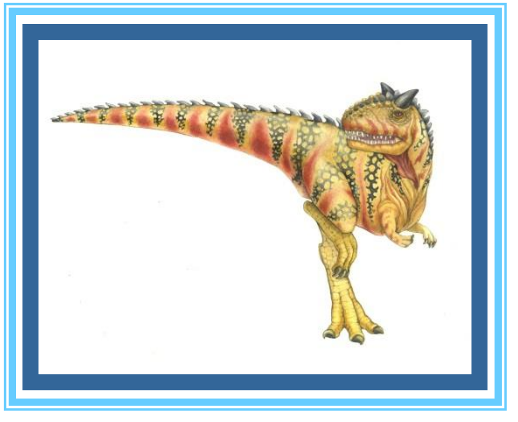
Chapter 3: Processes (II)



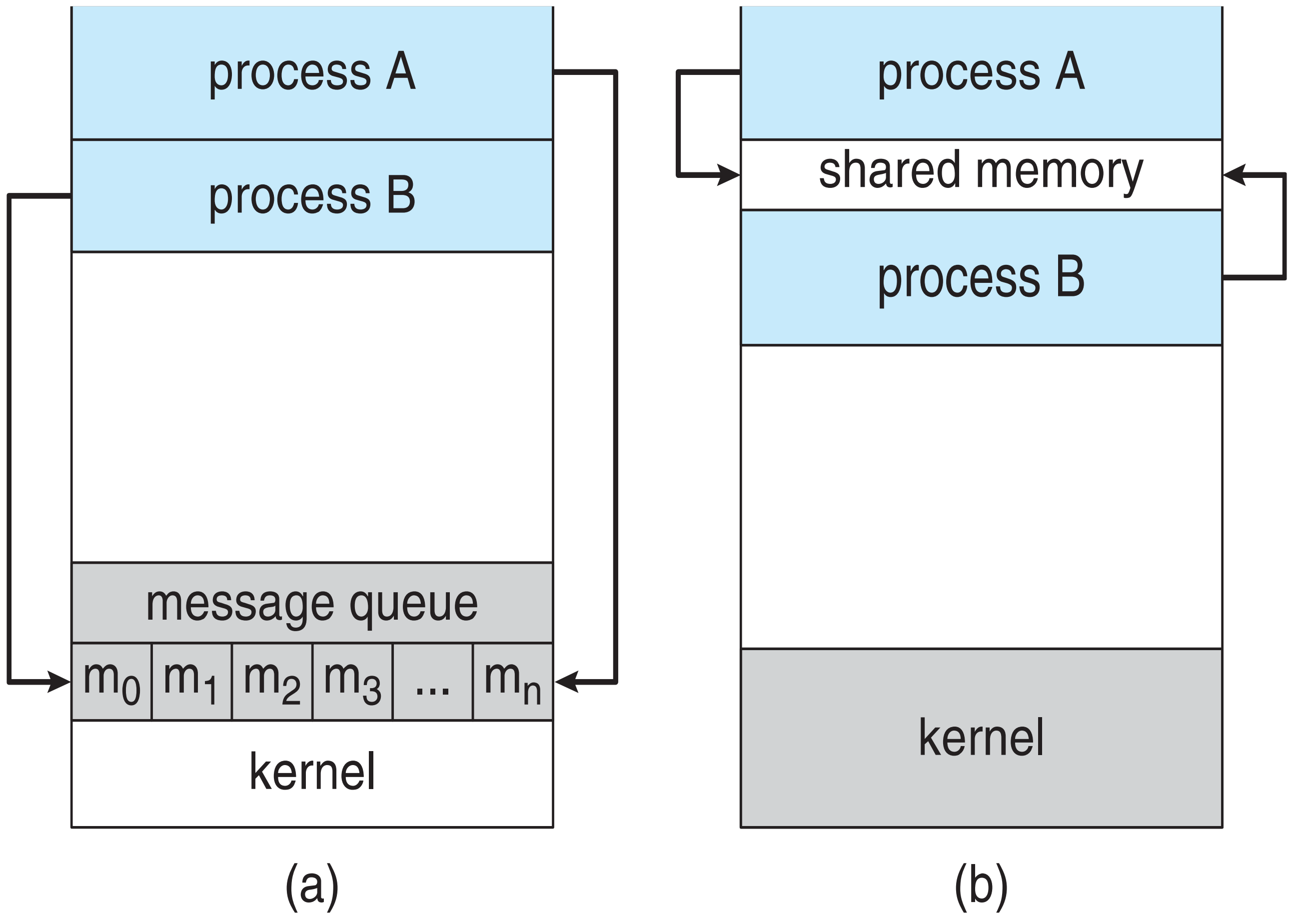
Interprocess Communication

- Processes within a system may be ***independent*** or ***cooperating***
- Cooperating process can affect or be affected by other processes, including sharing data

- Cooperating processes need **interprocess communication** (**IPC**)
- Two models of IPC
  - **Shared memory**
  - **Message passing**

Communications Models

(a) Message passing. (b) shared memory.



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
- Advantages of process cooperation
  - Information sharing : (for instance, a shared file),
  - Computation speed-up: If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others
  - Modularity: We may want to construct the system in a modular fashion, dividing the system functions into separate processes or threads.
  - Convenience: Even an individual user may work on many tasks at the same time

Producer-Consumer Problem

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process (EX: Compiler and assembler , Server and Client)

Inter-process Communication – Shared Memory

- An area of memory shared among the processes that wish to communicate
- The communication is under the control of the users processes not the operating system.
- Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.
- **Solving producer- consumer problem using shared memory**
  - **unbounded-buffer** places no practical limit on the size of the buffer
  - **bounded-buffer** assumes that there is a fixed buffer size

Interprocess Communication – Message Passing

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables (Chat program)
- IPC facility provides two operations:
  - **`send`**(*message*)
  - **`receive`**(*message*)
- The *message* size is either fixed or variable

Message Passing (Cont.)

- If processes *P* and *Q* wish to communicate, they need to:
  - Establish a ***communication link*** between them
  - Exchange messages via send/receive
- Implementation issues:
  - 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?

Message Passing (Cont.)

- Implementation of communication link
  - Physical:
    - Shared memory
    - Hardware bus
    - Network
  - Logical:
    - Direct or indirect
    - Synchronous or asynchronous
    - Automatic or explicit buffering

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
- Properties of communication link:
  - A link is established between a pair of processes only if both members of the pair have a shared mail box
  - A link may be associated with more than two processes
  - Between each pair of processes, there may be a number of different links, with each link corresponding to one mailbox

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.

Indirect Communication

- A mailbox may be owned either by a process or by the operating system.
  - If the mailbox is owned by a process (that is, the mailbox is part of the address space of the process)
  - When a process that owns a mailbox terminates, the mailbox disappears.
- In contrast, a mailbox that is owned by the operating system has an existence of its own.
- The process that creates a new mailbox is that mailbox’s owner by default.
- Initially, the owner is the only process that can receive messages through this mailbox. However, the ownership and receiving privilege may be passed to other processes through appropriate system calls

Synchronization

- Message passing may be either blocking or non-blocking
- **Blocking** is considered **synchronous**
  - **Blocking send** -- the sender is blocked until the message is received
  - **Blocking receive** -- the receiver is blocked until a message is available
- **Non-blocking** is considered **asynchronous**
  - **Non-blocking send** -- the sender sends the message and continue
  - **Non-blocking receive** -- the receiver receives:
    - A valid message, or
    - Null message
- Different combinations possible
  - If both send and receive are blocking, we have a **rendezvous**

Buffering

- Whether communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue
- Basically, such queues can be implemented in three ways:
  - **Zero capacity**. The queue has a maximum length of zero; thus, the link cannot have any messages waiting in it. In this case, the sender must block until the recipient receives the message.
  - **Bounded capacity**. The queue has finite length *n;* thus, at most *n* messages can reside in it. If the queue is not full when a new message is sent, the message is placed in the queue, and the sender can continue execution without waiting. The link's capacity is finite, however. If the link is full, the sender must block until space is available in the queue.
  - **Unbounded capacity**. The queue's length is potentially infinite; thus, any number of messages can wait in it. The sender never blocks.

End of Chapter 3