Inter Process Communication

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
  - Computation speed-up
  - Modularity
  - Convenience

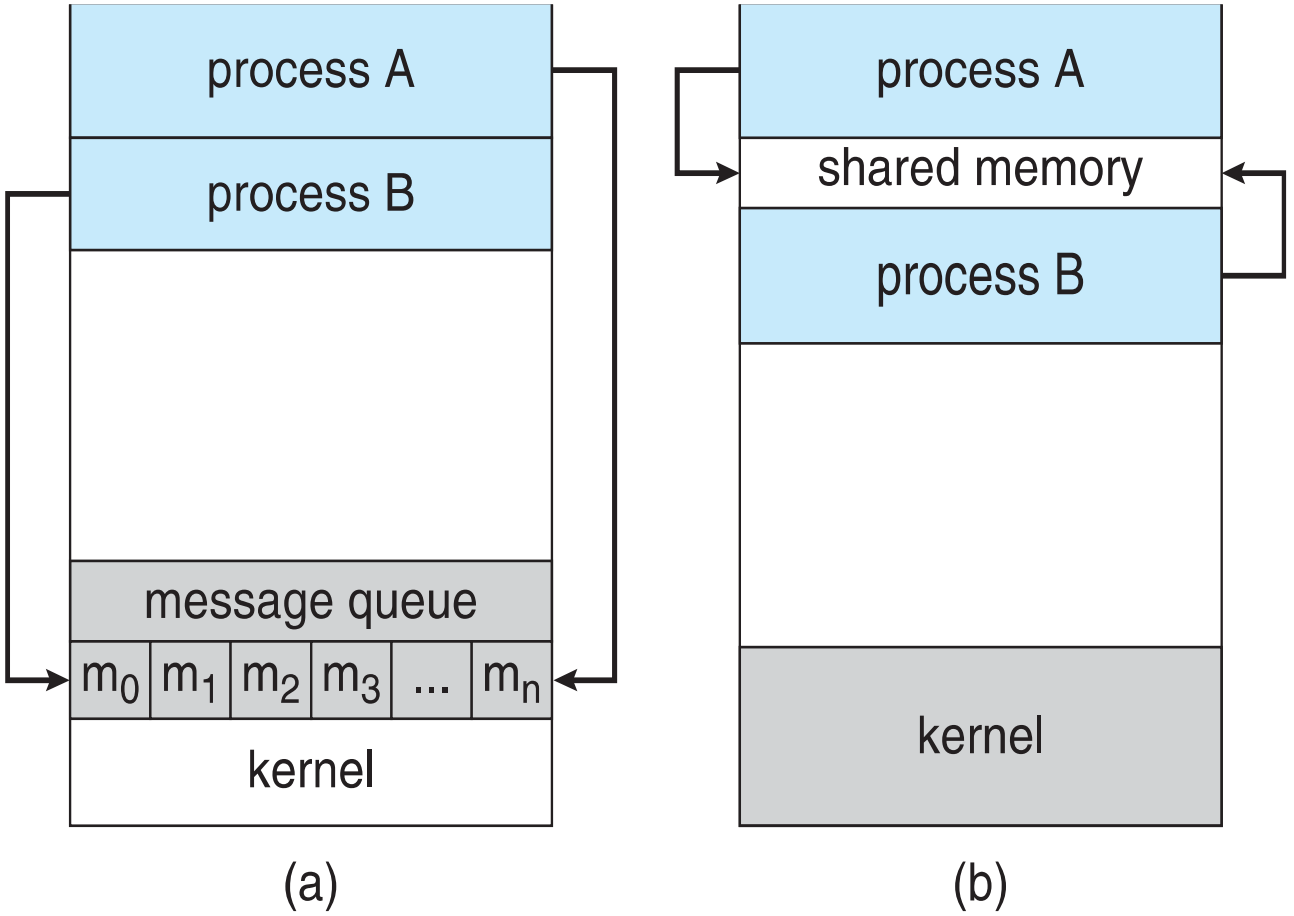
Inter process Communication

- Processes within a system may be ***independent*** or ***cooperating***
- Cooperating process can affect or be affected by other processes, including sharing data
- Inter Process communication is a mechanism which allows process to communicate each other and synchronize their actions.
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Resource sharing
  - Convenience
- Cooperating processes need **inter process communication** (**IPC**)
- Two models of IPC
  - **Shared memory**
  - **Message passing**

- Information sharing: Since some users may be interested in the same piece of information (for example, a shared file), you must provide a situation for allowing concurrent access to that information.
- Computation speedup: If you want a particular work to run fast, you must break it into sub-tasks where each of them will get executed in parallel with the other tasks. Note that such a speed-up can be attained only when the computer has compound or various processing elements like CPUs or I/O channels.
- Modularity: You may want to build the system in a modular way by dividing the system functions into split processes or threads.
- Convenience: Even a single user may work on many tasks at a time. For example, a user may be editing, formatting, printing, and compiling in parallel.

Communications Models

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



Shared Memory

- Communication between processes using shared memory requires processes to share some variable and it completely depends on how programmer will implement it.
- There are two processes: Producer and Consumer. Producer produces some item and Consumer consumes that item.
- The two processes shares a common space or memory location known as buffer where the item produced by Producer is stored and from where the Consumer consumes the item if needed.

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.

Producer-Consumer problem

- There are two processes: Producer and Consumer. Producer produces some item and Consumer consumes that item.
- The two processes share a common space or memory location known as a buffer where the item produced by Producer is stored and from which the Consumer consumes the item, if needed.

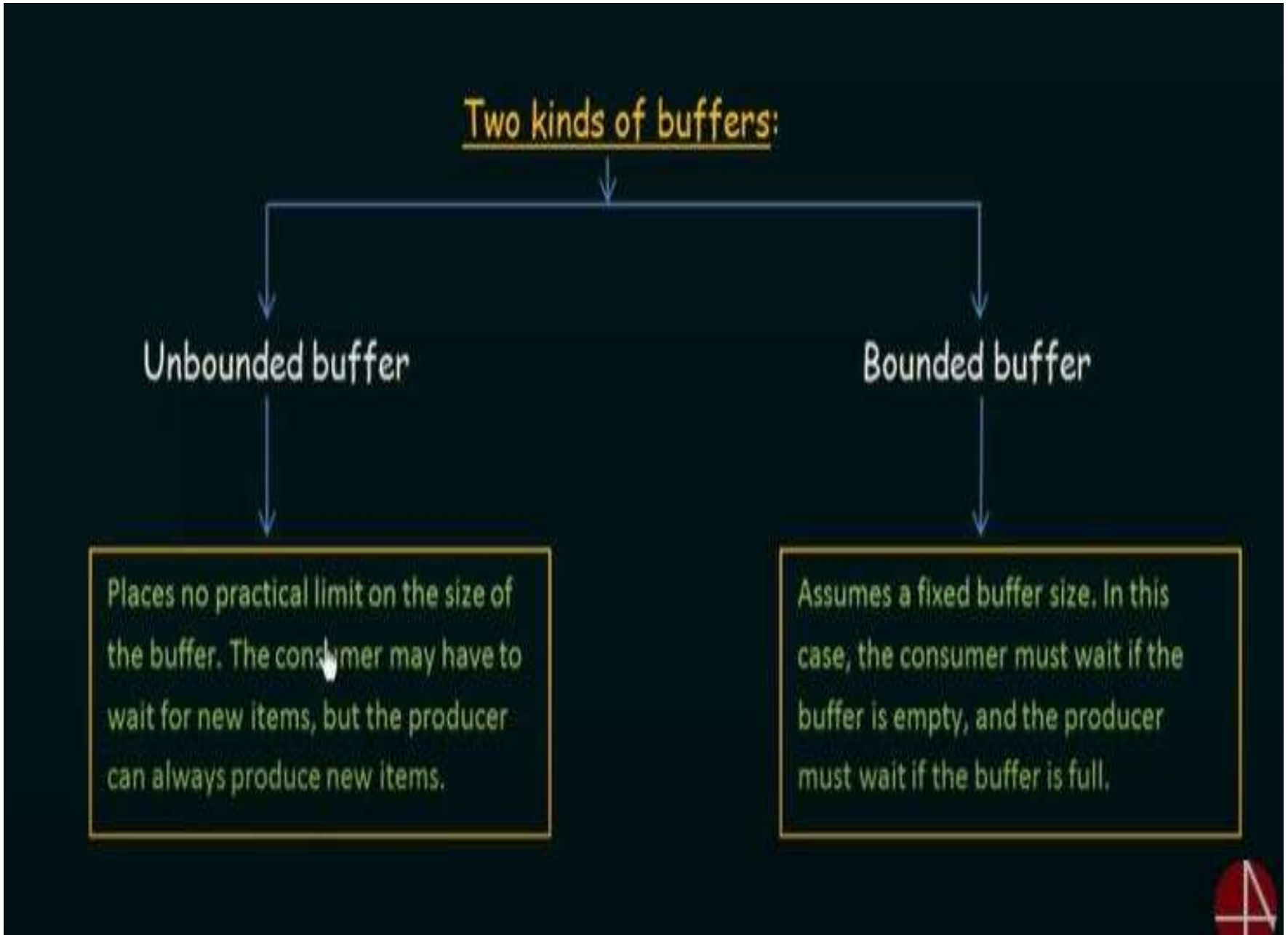
Producer Consumer Problem

A producer process produces information that is consumed by a consumer process.

For example, a compiler may produce assembly code, which is consumed by an assembler. The assembler, in turn, may produce object modules, which are consumed by the loader.

- One solution to the producer-consumer problem uses shared memory.
- To allow producer and consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by the consumer.

- One solution to the producer-consumer problem uses shared memory.
- To allow producer and consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by the consumer.
- This buffer will reside in a region of memory that is shared by the producer and consumer processes.
- A producer can produce one item while the consumer is consuming another item.
- The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

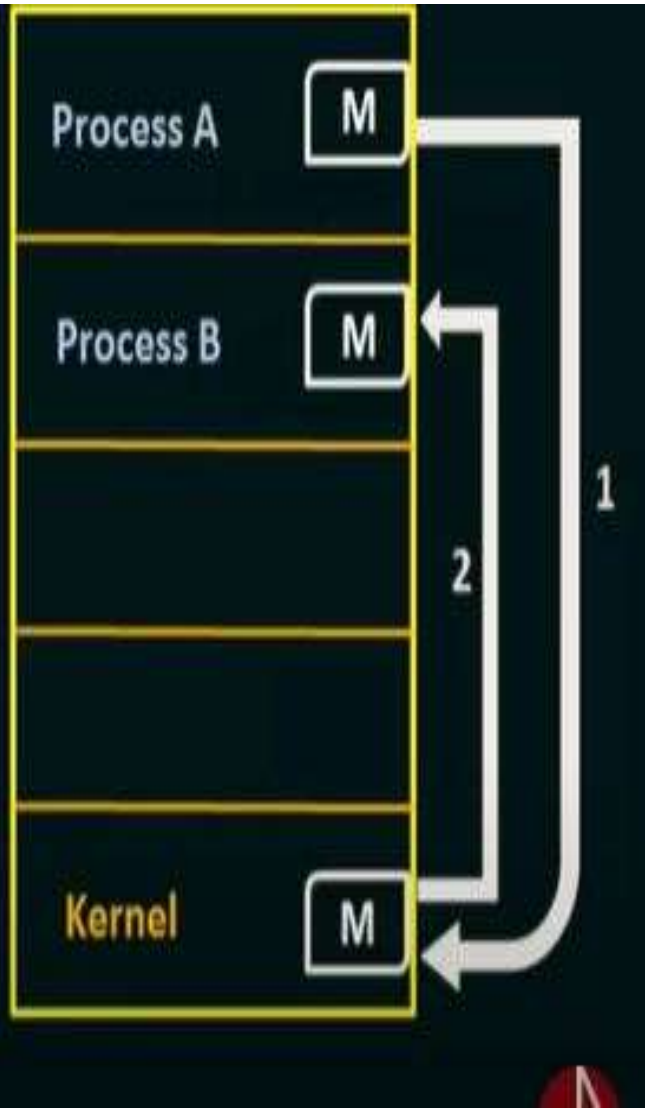

Two kinds of buffers:
Unbounded buffer
Bounded buffer
Places no practical limit on the size of the buffer. The consumer may have to wait for new items, but the producer can always produce new items.
Assumes a fixed buffer size. In this case, the consumer must wait if the buffer is empty, and the producer must wait if the buffer is full.

Inter process 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
- IPC facility provides two operations:
  - **`send`**(*message*)
  - **`receive`**(*message*)
- The *message* size is either fixed or variable

Inter process Communication – Message Passing

Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space and is particularly useful in a distributed environment, where the communicating processes may reside on different computers connected by a network.



A message-passing facility provides at least two operations:

- send (message)

and

- receive (message)

Messages sent by a process can be of either fixed or variable size.

FIXED SIZE: The system-level implementation is straightforward.
But makes the task of programming more difficult.

VARIABLE SIZE: Requires a more complex system-level implementation.
But the programming task becomes simpler.

Message-Passing Systems (Part-2)

If processes P and Q want to communicate, they must send messages to and receive messages from each other.

A communication link must exist between them.

This link can be implemented in a variety of ways. There are several methods for logically implementing a link and the send( ) /receive( ) operations, like:

- Direct or indirect communication
- Synchronous or asynchronous communication
- Automatic or explicit buffering

There are several issues related with features like:

- Naming
- Synchronization
- Buffering

Naming

Processes that want to communicate must have a way to refer to each other.

They can use either **direct** or **indirect** communication.

Under direct communication- Each process that wants to communicate must explicitly name the recipient or sender of the communication.

- send (P, message) - Send a message to process P.
- receive (Q, message) - Receive a message from process Q.

A communication link in this scheme has the following properties:

- A link is established automatically between every pair of processes that want to communicate. The processes need to know only each other's identity to communicate.
- A link is associated with exactly two processes.
- Between each pair of processes, there exists exactly one link.

Another variant of Direct Communication- Here, only the sender names the recipient; the recipient is not required to name the sender.

- send (P, message) - Send a message to process P.
- receive (id, message) - Receive a message from any process; *the variable id is set to the name of the process with which communication has taken place.*

This scheme employs **asymmetry in addressing.**

The disadvantage in both of these schemes (symmetric and asymmetric) is the limited modularity of the resulting process definitions.

Changing the identifier of a process may necessitate examining all other process definitions.

With indirect communication:

The messages are sent to and received from **mailboxes**, or ports.

- A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed.
- Each mailbox has a unique identification.
- Two processes can communicate only if the processes have a shared mailbox

  - send (A, message) — Send a message to mailbox A.
  - receive (A, message) — Receive a message from mailbox A.

A communication link in this scheme has the following properties:

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

A communication link in this scheme has the following properties:

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

Synchronization

Communication between processes takes place through calls to send() and receive () primitives. There are different design options for implementing each primitive.

Message passing may be either **blocking** or **nonblocking**— also known as **synchronous** and **asynchronous**.

Message passing may be either **blocking** or **nonblocking**— also known as **synchronous** and **asynchronous**.

Blocking send: The sending process is blocked until the message is received by the receiving process or by the mailbox.

Nonblocking send: The sending process sends the message and resumes operation.

Blocking receive: The receiver blocks until a message is available.

Nonblocking receive: The receiver retrieves either a valid message or a null.

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

- All messaging system require the framework to temporarily buffer messages.
- Queue of messages attached to the link.
- implemented in one of three ways
  1. Zero capacity – no messages are queued on a link.
     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