SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NAAC-UGC with 'A' Grade

Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE

Course Code and Name : 23ITB203 / Principles of Operating Systems

II YEAR / IV SEMESTER

Unit 2:Inter Process Communication

Interprocess communication

Processes executing concurrently in the operating system may be either independent processes or cooperating processes.

Independent processes - They cannot affect or be affected by the other processes executing in the system.

Cooperating processes - They can affect or affected by the other processes executing in the system.

Any process that shares the data with other processes is a cooperating process.

There are several reasons for providing an environment that allows process cooperation:

Information sharing

Computation speedup

Modularity

Convenience

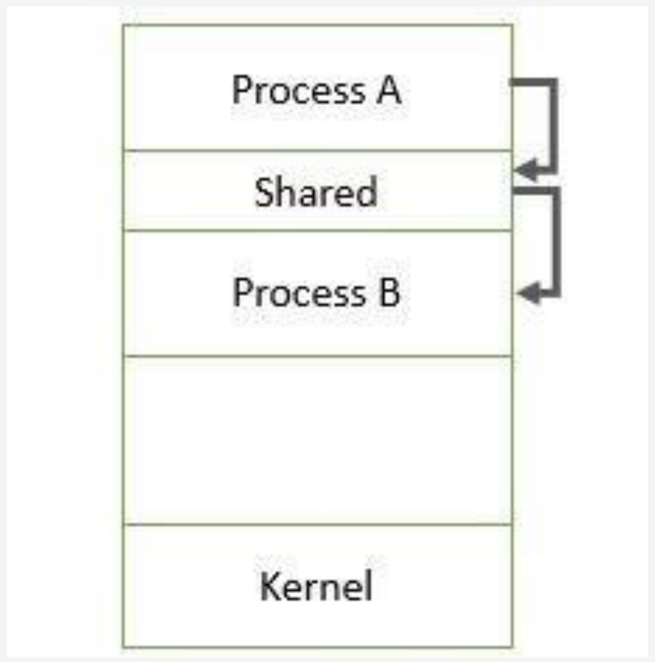
Cooperating processes require an Interprocess communication (IPC mechanism) that will allow them to exchange data and information.

There are two fundamental models of Interprocess communication

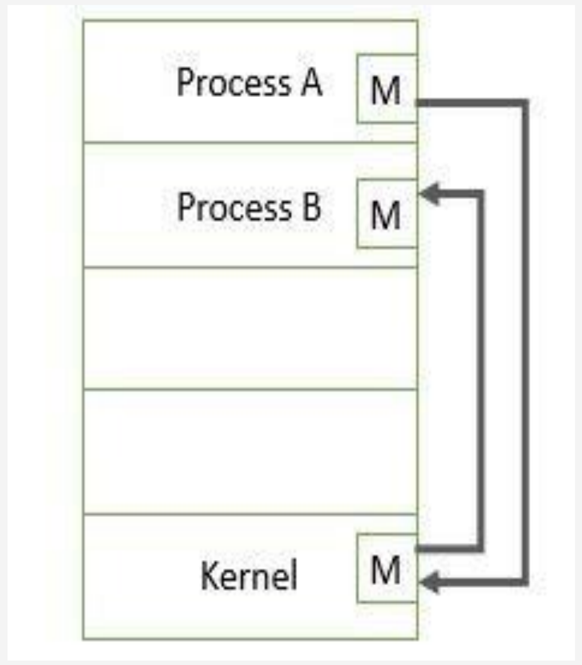
(1) Shared memory

(2) Message passing

- In the shared memory model, a region of memory that is shared by cooperating processes is established. Processes can exchange information by reading or writing data to the shared region.
- In the message passing model, communication takes place by means of messages exchanged between the cooperating processes.



(a)

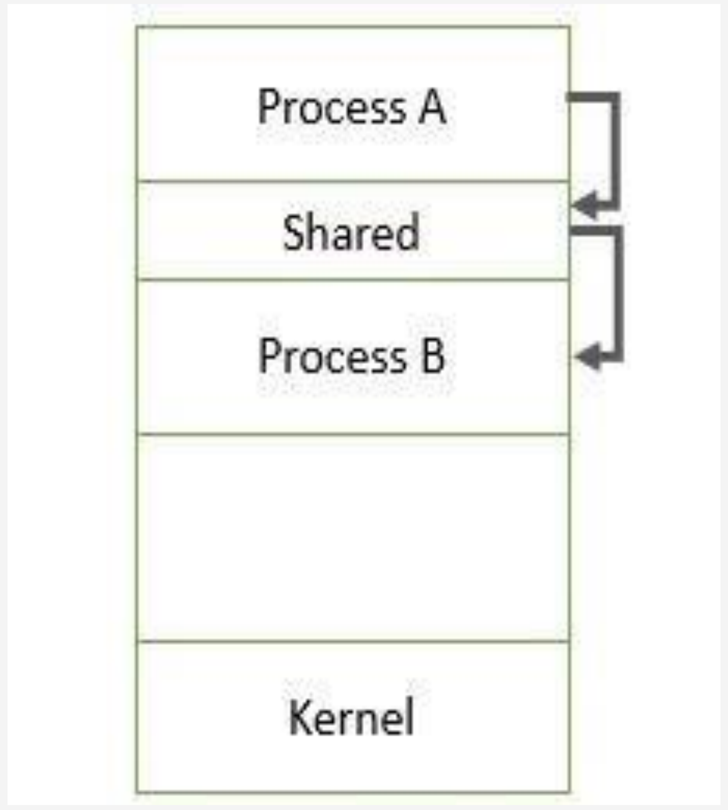


(b)

Fig: Communication models, (a) Shared memory, (b) Message passing

Shared Memory Systems

- Interprocess communication using shared memory requires communicating processes to establish a region of shared memory
- Typically a shared-memory region resides in the address space of the process creating the shared-memory segment.
- Other processes that wish to communicate using this shared-memory segment must attach it to their address space.
- Normally, the operating system tries to prevent one process from accessing another process's memory
- Shared memory requires that two or more processes agree to remove this restriction.



Producer consumer problem

A producer 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.
- 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 one.
- 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 buffer

Unbounded buffer

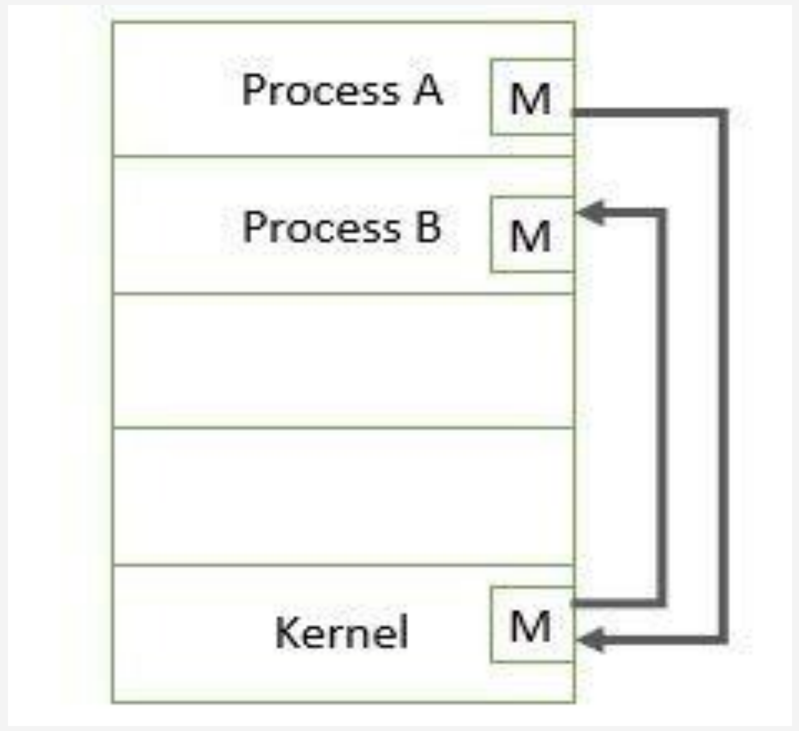
Places no practical limit on the size of the buffer. The consumer may have to wait for few new items, but the producer can always produce new items.

Bounded buffer

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.

Message-Passing Systems (Part-1)

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)

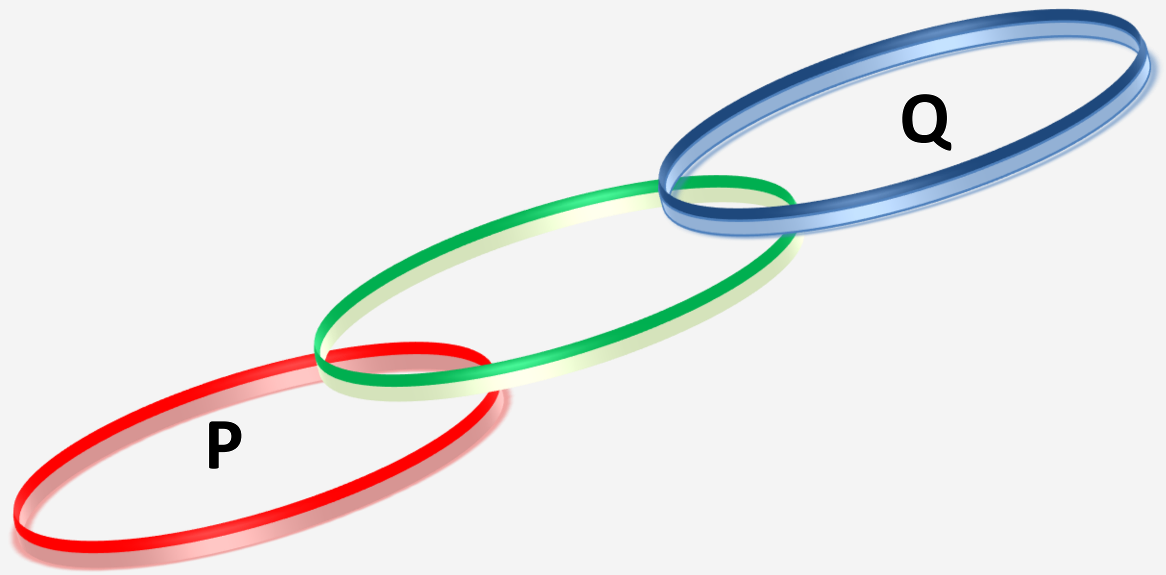
  and

- receive (message)

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

Fixed size:	The system-level implementation is straight forward. but makes the task of programming more difficult.
Variable size:	Requires a more complex system-level implementation. but the programming task become simpler.

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



- Direct or indirect communication
- Synchronous / 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.

This scheme exhibits **Symmetry in addressing**; that is, both the sender process and the receiver process must name the other to communicate

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 linear 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.

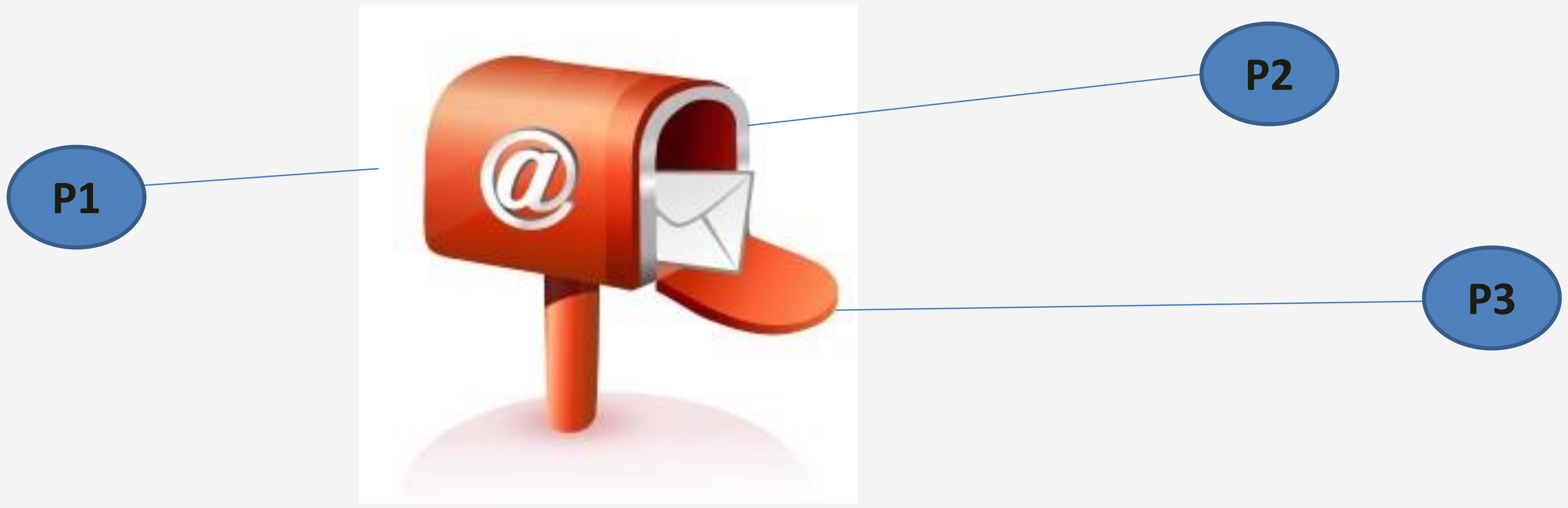
- A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed.
- Each mailbox has a unique identification
- Two processes can communicate only if the processes have a shared mailbox.

  - 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 communication processes, there may be a number of different links, with each link corresponding to one mailbox.

Now suppose that processes P1, P2 and P3 all share mailbox A



Process P1 sends a message to A, while both P2 and P3
execute a receive() from A. Which process will receive the message sent by P1?

The answer depends on which of the following methods we choose:

- Allow a link to be associated with two process at most.
- Allow at most one process at a time to execute a receive() operation.
- Allow the system to select arbitrarily which process will receive the message (that is, either P2 or
- P3, but not both, will receive the message). The system also may define an algorithm for selecting which process will receive the message(that is, round robin where processes take turns receiving the messages). The system may identify the receiver to the sender.

A **mailbox** may be **owned** either by a **process** or by the **operating system**.

Synchronization

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

Messages 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.

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

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

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

Buffering

Whether communication is direct or indirect, messages exchanged by communicating processes reside in a 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 message 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 utmost 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 links capacity is finite, however. If the link is full, the sender must block until space is available in the queue.

Unbounded capacity: The queue length is potentially infinite; thus, any number of messages can wait in it. The sender never blocks.

Thank You