2.4 INTERPROCESS COMMUNICATION

- A process is independent if it cannot affect or be affected by the other processes executing in the system. Any process that does not share data with any other process is independent.
- A process is cooperating if it can affect or be affected by the other processes executing in the system. Clearly, any process that shares data with other processes is a cooperating process.
- Advantages of cooperating process

  i) Information sharing

  ii) Computation speedup

  iii) Modularity

  iv) Convenience**.**

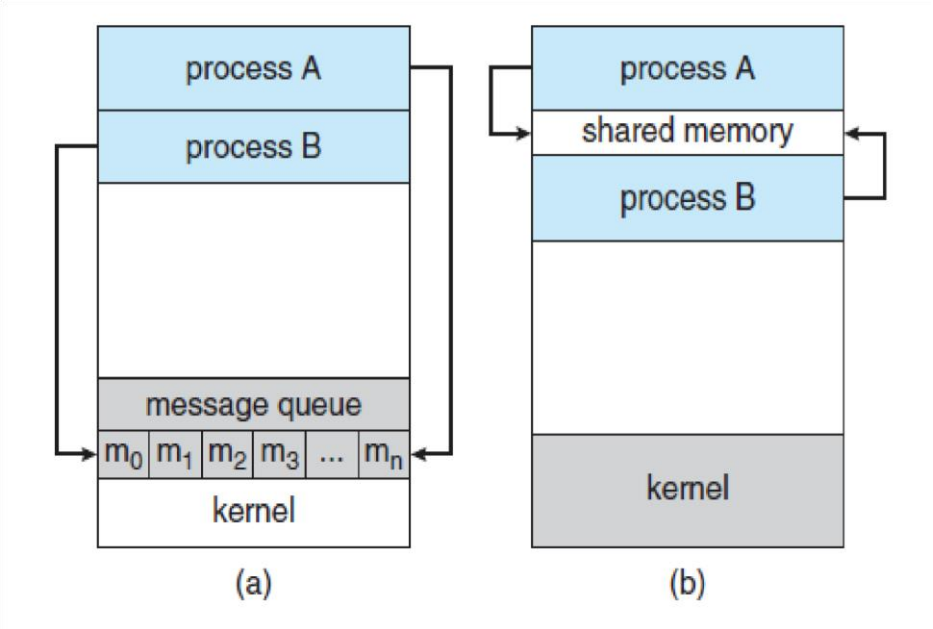
DEFINITION:

An Inter process communication is a mechanism that allows the cooperating process to exchange data and communication among each other.

There are two fundamental models of Inter process communication

- **Shared Memory model**
- **Message passing model**

- In shared memory model a region of memory is shared by the cooperating process. Processes can then exchange information by reading and writing data to the shared region.
- In the message-passing model, communication takes place by means of messages exchanged between the cooperating processes.
- Message passing is easier to implement in a distributed system than shared memory.
- The shared memory is faster than that of message passing.



Communications models: (a) Message passing. (b) Shared memory.

2.4.1 Shared-Memory Systems

- Inter process communication using shared memory requires communicating processes to establish a region of shared memory.
- 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. They can then exchange information by reading and writing data in the shared areas.

EXAMPLE: PRODUCER – CONSUMER PROCESS:

- A **producer** process produces information that is consumed by a **consumer** process.
- 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.
- Two types of buffers can be used.

- Bounded Buffer.
- Unbounded Buffer.
- The **unbounded 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.
- The **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.

CODE FOR PRODUCER PROCESS:

```
item next produced;
while (true) {
/* produce an item in next produced */
while (((in + 1) % BUFFER SIZE) == out);
/* do nothing */
buffer[in] = next produced;
in = (in + 1) % BUFFER SIZE;
}
```

The producer process has local variable next produced in which the new item to be produced is stored.

The consumer process has a local variable next consumed in which the item to be consumed is stored.

This scheme allows at most BUFFER SIZE − 1 items in the buffer at the same time.

CODE FOR CONSUMER PROCESS

```
item next consumed;
while (true) {
while (in == out); /* do nothing */
/* Get the next available item */
nextConsumed = buffer[ out ];
out = ( out + 1 ) % BUFFER_SIZE;
}
```

2.4.2 Message-Passing Systems

- Message passing systems must support at a minimum system calls for "send message" and "receive message".
- A communication link must be established between the cooperating processes before messages can be sent.
- There are three key issues to be resolved in message passing systems
- Direct or indirect communication (naming)
- Synchronous or asynchronous communication
- Automatic or explicit buffering.

2.4.2.1 Naming

Each process that wants to communicate must explicitly name the recipient or sender of the communication. Direct communication can be done in two ways symmetric addressing and asymmetric addressing.

- With **direct communication** the sender must know the name of the receiver to which it wishes to send a message.
- There is a one-to-one link between every sender-receiver pair.
- For **symmetric** communication, the receiver must also know the specific name of the sender from which it wishes to receive messages.

  send(P, message)—Send a message to process P.

  receive(Q, message)—Receive a message from process Q.
- For **asymmetric** communications, this is not necessary.

  send(P, message)—Send a message to process P.

  receive(id, message)—Receive a message from any process
- **Indirect communication** uses shared mailboxes, or ports.

  send(A, message)—Send a message to mailbox A.

  receive(A, message)—Receive a message from mailbox A.
- Multiple processes can share the same mailbox or boxes.
- Only one process can read any given message in a mailbox. Initially the process that creates the mailbox is the owner, and is the only one allowed to read mail in the mailbox, although this privilege may be transferred.

The OS must provide system calls to create and delete mailboxes, and to send and receive messages to/from mailboxes.

3.4.2.2 Synchronization

Either the sending or receiving of messages (or neither or both) may be either **blocking** or **non-blocking**.

- 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.

3.4.2.3 Buffering

Messages are passed via queues, which may have one of three capacity configurations:

1. **Zero capacity** - Messages cannot be stored in the queue, so senders must block until receivers accept the messages.
2. **Bounded capacity**- There is a certain pre-determined finite capacity in the queue. Senders must block if the queue is full, until space becomes available in the queue, but may be either blocking or non-blocking otherwise.
3. **Unbounded capacity** - The queue has a theoretical infinite capacity, so senders are never forced to block.