**INTER-PROCESS COMMUNICATION**

* Process executing concurrently in an O.S. may be either independent or cooperating processes. A process is independent if it cannot affect or be affected by the processes executing in the system.
* Any process that does not share data with any other process in independent.
* A process is cooperating if it can affect or can be affected by the processes executing in the system.
* There are several reasons for providing an environment that allow process cooperation.

1. INFORMATION SHARING

Several users may want to access single piece of information; we must provide an environment to allow concurrent access to such information.

1. COMPUTATION SPEED UP

If we want a particular task to run faster we must break it into sub tasks, each of which will be executing in parallel with others. It can only be achieved if the computer has multiple processing element.

1. MODULARITY

If we want to construct the system in modular fashion, dividing the system function into separate processes or making of threads.

1. CONVINIENCE

Even an individual user may work on many tasks at the same time. Cooperating processes require inter-process communication (IPC) mechanism that will allow them to exchange data and information.

There are two fundamental modes inter-process communication.

* Shared Memory
* Message Sharing

**SHARED MEMORY**

A region of memory is shared by cooperating processes is established in this model. Processes can then exchange information by reading and writing data to the shared region. Typically, a shared memory region resides in the address space of the process creating the shared memory segment. Operating System tries to prevent one process from exiting another processes memory. Shared memory requires that two or more processes agrees to remove this restriction then they can exchange information by reading and writing data in the shared areas.

The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

**Example:** Let us consider a producer consumer problem which is a common paradigm for cooperating processes. A producer process produces information that is consumed by a consumer process.

**Example:** A print program produces characters that are consumed by the printer driver, a compiler produce assembly code consumed by the assembler. The assembler in turn, produce object module which are consumed by the loader.

To allow the consumer and customer processes to run concurrently we must have an available buffer of items that can be filled by the producer and emptied by consumer. A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized so that the customer does not try to consume the item that has not been produced yet.

**MESSAGE PASSING SYSTEMS**

The function of this system is to allow processes to communicate with one another without the need to resort to shared data. In this scheme, services are provided as ordinary user processes. Communication among user processes is accomplished through the passing of messages. An IPC facility provides at least two operations send and receive messages. Messages send by a process can be either fixed or variable sized.

Several methods on logically implementing a link and the send/ receive operations are:

1. Direct or indirect communication
2. Symmetric or asymmetric communication
3. Automatic or explicit buffering
4. Send by copy or send by reference
5. Fixed size or variable size messages

**NAMING**

Processes that want to communicate must have a way to refer to each other they can use direct or indirect communication.

1. DIRECT COMMUNICATION

Each process that want to communicate must explicitly name the recipient or centre of the communication. In this theme send and receive primitive are defined as

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

Communication link in this scheme has the following properties:

1. A link is established between every pair of processes that want to communicate.

The processes need to know only each other identify to communicate.

1. A link is associated with exactly two process.
2. Exactly one link exists between each pair of processes.

This scheme follows symmetry in addressing i.e. both sender and receiver processes must name the other to communicate.

A variant of this scheme employs asymmetry in addressing. Only the sender name, a recipient; A recipient is not required to name the sender in this scheme send and receive primitive are as follows:

1. Send (P, message): send a message to process P.
2. Receiver (ID, message): receiver a message from any process; a variable ID is given to the process with which the communication has taken place.

The disadvantage of both symmetric and asymmetric schemes is the limited modularity of resulting process definition.

1. INDIRECT COMMUNICATION

With indirect communication the messages are sent to and received from mail boxes or ports. A mail box is an object in which the messages can be placed by processes and from which the messages can be removed. Each mail box has a unique identification. A process can communicate with some other process via a number of different mail boxes.

Two processes can communicate only if they share a mail box. A send/ receive primitive are as follows:

1. Send (A, message): send a message to mail box A.
2. Receive (A, message): receive a message from mail box A.

The communication link has the following properties:

1. A link may be associated with more than two processes.
2. A number of different lanes may exist between each pair of communicating processes, with each link corresponding to one mail box.

**Question:** If processes P1, P2, P3 share mail box A. process P1 sends the message to range A, while P2 and P3 each exchange a receive from A. which process will receive the message sent by P1.

**Answer:** It depends on the scheme that we choose:

1. Allow a link to be associated with at most two processes.
2. Allow at most one process at a time to execute a receive operation.
3. Allow the system to select arbitrarily which process will receive the message. The system may identify the receiver to the sender.
4. A mail box owned by a O.S. is independent and is not attached to any particular process. The O.S. then must provide a mechanism that allow a process to do the following.

* Create a new mail box.
* Send and receive messages through the mail box.
* Delete a mail box.

**SYNCHRONIZATION**

* Communication between processes takes place by calls to send and receive primitive.
* Message passing may be either blocking or non-blocking also known as synchronous and non-synchronous respectively.

1. Blocking Send:

sending process is blocked until the message is received by the receiving process or by mail box.

1. Non-Blocking Send:

sending process sends the message and resumes operations.

1. Blocking Receive:

the receiver blocks until a message is available.

1. Non-Blocking Receiver:

the receiver retrieves a valid message or a null.

**BUFFERING**

Whether the communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue. Such a queue can be implemented in three ways:

1. Zero Capacity:

queue has maximum length zero thus, the link cannot have any messages waiting in it. It is sometimes referred to as a message system with no buffering. Sender must block until the recipient receives the message.

1. Bounded Capacity:

the queue has finite length ‘n’ thus, most ‘n’ messages can reside in it. If the queue is not full, when a new message is sent, the latter is placed in the queue and the sender can continue execution without waiting. It is referred to as automatically.

1. Unbounded Capacity:

The queue has potentially infinite length thus any number of messages can wait in it. The sender never blocks. It is also referred to as automatic buffering.