OPERATING SYSTEMS

Interprocess Communication - IPC

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

OPERATING SYSTEMS

- ➢ InterProcess Communication
  - ■ Shared Memory
  - ■ Messages
  - ■ Named and Unnamed Pipes

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

OPERATING SYSTEMS

Scheduling Criteria

Topics to be covered	% of Portions covered	
	Reference chapter	Cumulative
Introduction: What Operating Systems Do, Computer-System Organization	1.1, 1.2	21.4
Computer-System Architecture, Operating-System Structure & Operations	1.3,1.4,1.5	
Kernel Data Structures, Computing Environments	1.10, 1.11	
Operating-System Services, Operating-System Design and Implementation	2.1, 2.6	
Process concept: Process in memory, Process State, Process Control Block, Context switch, Process Creation & Termination,	3.1 – 3.3	
CPU Scheduling - Preemptive and Non-Preemptive, Scheduling Criteria, FIFO Algrorithm	5.1-5.2	
Scheduling Algorithms:SJF, Round-Robin and Priority Scheduling	5.3	
Multi-Level Queue, Multi-Level Feedback Queue	5.3	
Multiprocessor and Real Time Scheduling	5.5, 5.6	
Case Study: Linux/ Windows Scheduling Policies.	5.7	
Inter Process Communication – Shared Memory, Messages	3.4	
Named and unnamed pipes (+Review)	3.6.3	

Basic Inter-Process Communication Concepts

- Processes within a system may be ***independent*** or ***cooperating***
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need **inter-process communication** (IPC)
- Two models of IPC
  - **Shared memory**
  - **Message passing**

Basic Inter-Process Communication Concepts

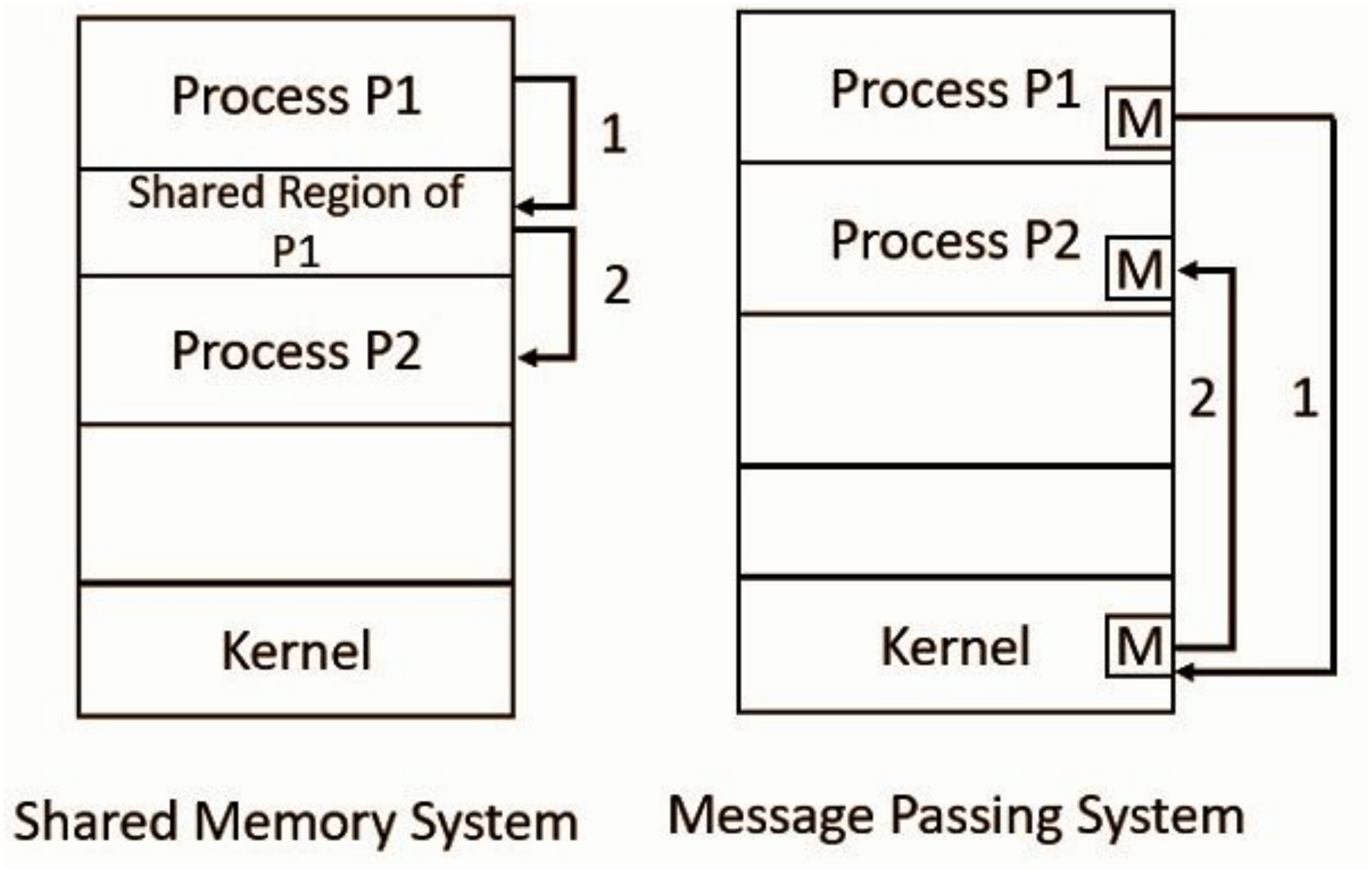
Inter Process Communication (IPC) is a mechanism that involves communication of one process with another process. This usually occurs only in one system and also be extended as communication between two process on two different hosts

Communication can be of two types –

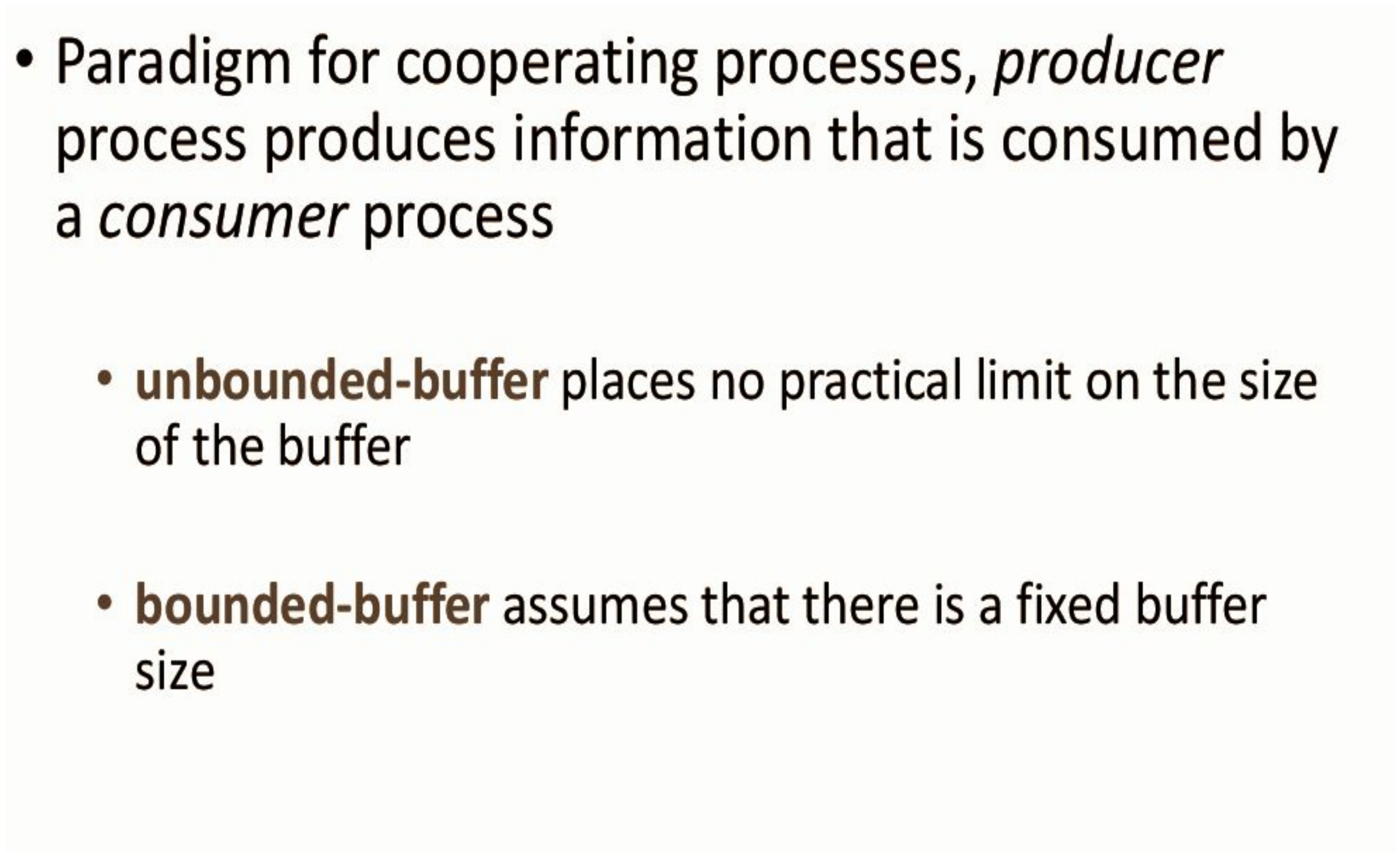
- Between related processes initiating from only one process, such as parent and child processes.

- Between unrelated processes, or two or more different processes.

Basic Inter-Process Communication Concepts



Basic Inter-Process Communication Concepts



- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process
  - **unbounded-buffer** places no practical limit on the size of the buffer
  - **bounded-buffer** assumes that there is a fixed buffer size

IPC - Producer Consumer Problem

- The Producer-Consumer problem is a classic problem this is used for multi-process synchronization i.e. synchronization between more than one processes. This also demonstrates the concept of Interprocess communication

- In the producer-consumer problem, there is one Producer process that is producing something and there is one Consumer process that is consuming the products produced by the Producer. The producers and consumers share the same memory buffer that is of fixed-size (bounded) or unlimited buffer size (Unbounded)

- The job of the Producer is to generate the data, put it into the buffer, and again start generating data. While the job of the Consumer is to consume the data from the buffer.

IPC - Producer Consumer Problem - Bounded Buffer - Shared Memory Solution

- Shared data

```
#define BUFFER_SIZE 10
typedef struct {
   . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- Solution is correct, but can only use BUFFER_SIZE-1 elements

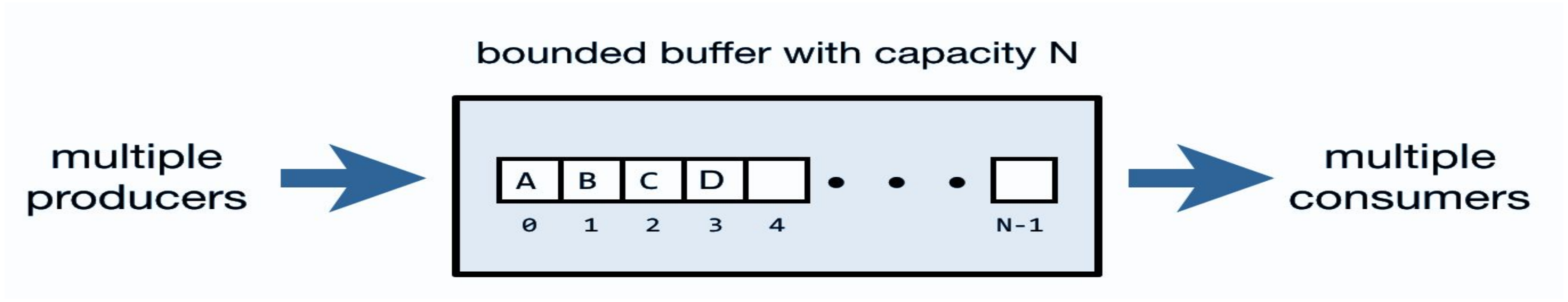
OPERATING SYSTEMS

IPC - Producer Consumer Problem - Bounded Buffer - Shared Memory Solution

```
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;
}

item next_consumed;
while (true) {
        while (in == out)
                ; /* do nothing */
        next_consumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;

        /* consume the item in next consumed */
}
```



IPC - Shared Memory

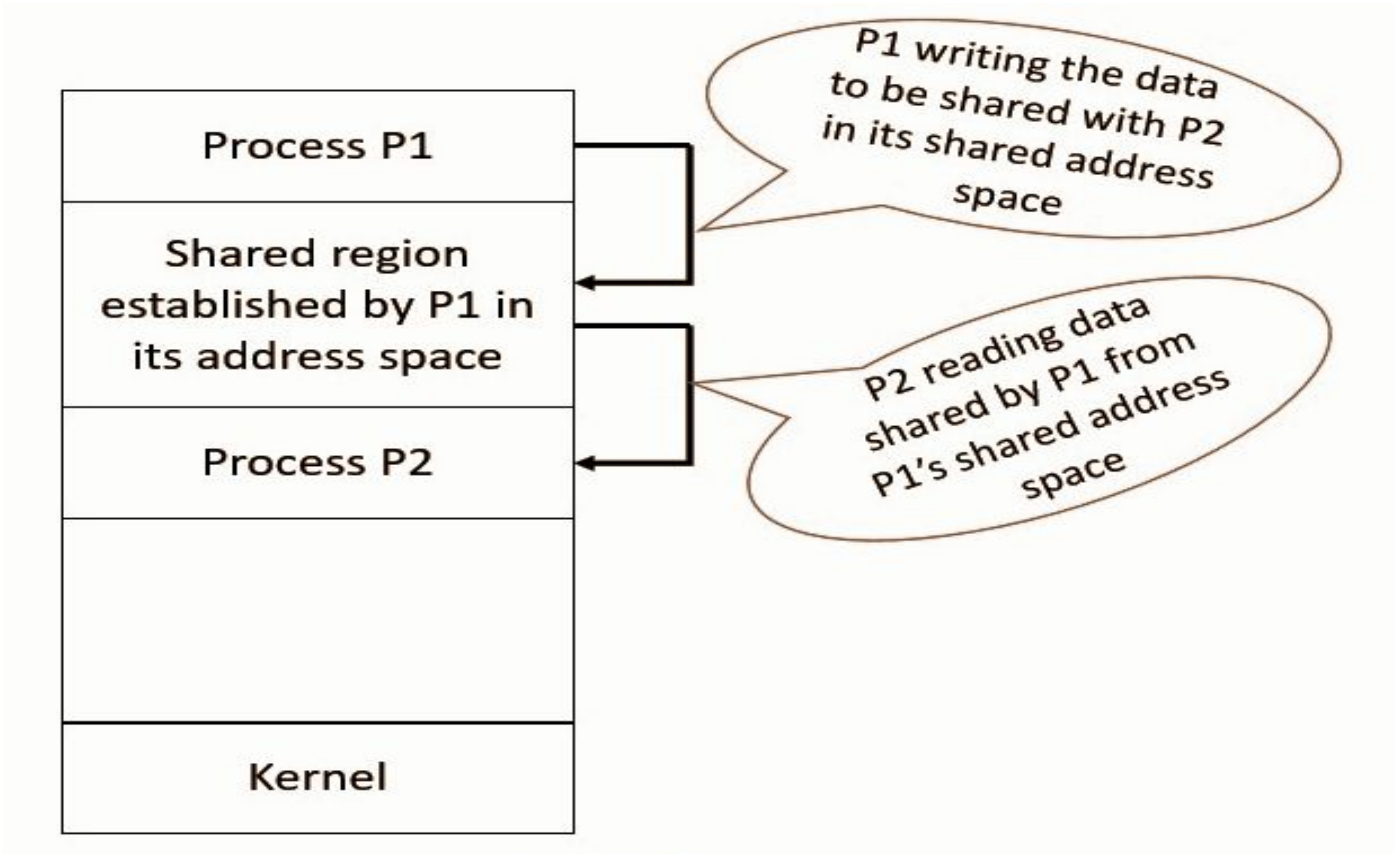
- Shared memory system is one of the fundamental models of **interprocess communication**. In the shared memory system, the cooperating processes communicate with each other by establishing the **shared memory** region, in its address space.

- Shared memory model allows the fastest interprocess communication.

- Among the cooperating processes, the process that wants to initiate the communication and have some data to share, establish the shared memory region in its address space. Next, another process that wishes to communicate and want to read the shared data, must attach itself to the initiating process's shared address space.

- The implementation needs a proper synchronisation mechanism, which is dealt later in the course in the chapter Process Synchronisation

IPC - Shared Memory


Process P1
Shared region established by P1 in its address space
Process P2
Kernel
P1 writing the data to be shared with P2 in its shared address space
P2 reading data shared by P1 from P1's shared address space

Shared Memory System

Advantages of Shared Memory

- Shared memory system is faster interprocess communication model.
- Shared memory allows cooperating processes to access the same pieces of data concurrently.
- Using shared memory, also speed ups the computation power of the system as the long task can be divided into smaller sub-tasks and can be executed in parallel.
- Modularity can also be achieved if we divide up the system functions into separate processes or threads.
- Using shared memory user can perform multiple tasks at a time. Like user can print, create a document, playing MP3 at the same time.

IPC - 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

What is Message Passing in Interprocess Communication ?

- Node => Machine=>Computer System=>OS=>Homogeneous, Heterogenous,Hybrid

- It refers to means of communication between
  - Different thread with in a process .
  - Different processes running on same node.
  - Different processes running on different node.

IPC - Message Passing

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?

IPC - Message Passing

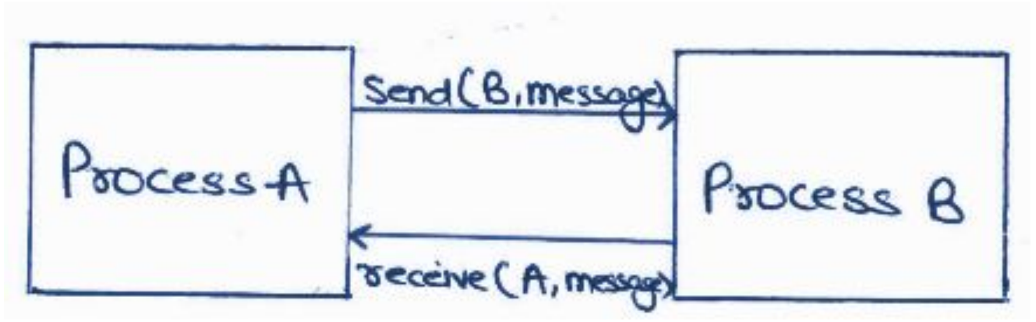
- Implementation of communication link
  - Physical:
    - Shared memory
    - Hardware bus
    - Network
  - Logical:
    - Direct or indirect
    - Synchronous or asynchronous
    - Automatic or explicit buffering

IPC - Message Passing - 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

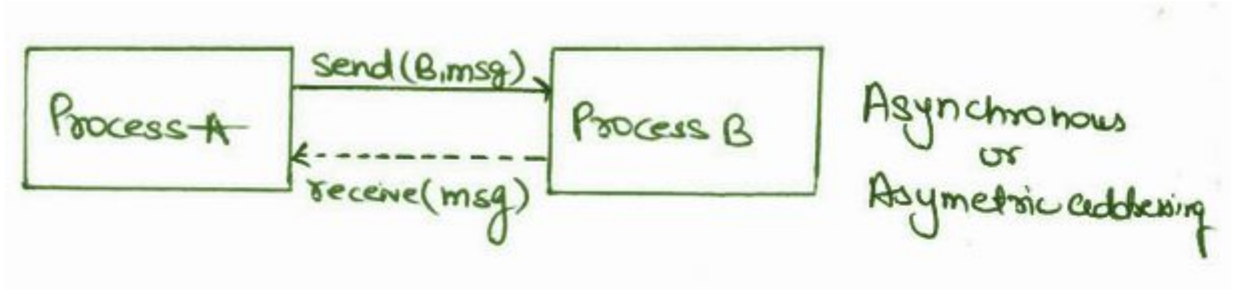
IPC - Message Passing - Direct Communication - Symmetric Addressing

- In this type that two processes need to name other to communicate. This becomes easy if they have the same parent.
- If process A sends a message to process B, then
  - send(B, message);
  - Receive(A, message);
- By message passing a link is established between A and B.
- Here the receiver knows the Identity of sender message destination. This type of arrangement in direct communication is known as Symmetric Addressing.



IPC - Message Passing - Direct Communication - Asymmetric Addressing

- Another type of addressing known as asymmetric addressing where receiver is not aware of the ID of the sending process in advance.



IPC - Message Passing - 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

IPC - Message Passing - Indirect Communication

- Operations
  - create a new mailbox (port)
  - 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

IPC - Message Passing - 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

IPC - Message Passing - Indirect Communication

- The following types of communication link are possible through mailbox.

    - One to One link: one sender wants to communicate with one receiver. Then single link is established.

    - Many to Many link: Multiple Sender want to communicate with single receiver.Example in client server system, there are many client processes and one server process. The mailbox here is known as PORT.

    - One to Many link: One sender wants to communicate with multiple receiver, that is to broadcast message.

    - Many to many: Multiple sender want to communicate with multiple receivers.

IPC - Message Passing - Indirect Communication - Synchronization

- Message passing may be either blocking or non-blocking.
- Blocking is considered synchronous
- Non-blocking is considered asynchronous
- send and receive primitives may be either blocking or non-blocking

IPC - Message Passing - Indirect Communication - Synchronization

- Synchronous receive
  - Receiving process blocks until message is copied into user-level buffer by the sender

- Asynchronous receive
  - Receiving process issues a receive operation (specifying a buffer) and then carries on with other tasks
  - It either polls the OS to find out if the receive has completed or gets an interrupt when the receive has completed
  - Threads (discussed later) allow you to program an asynchronous receive in a synchronous way
  - Issue a synchronous receive with one thread while carrying out other tasks with other threads

IPC - Message Passing - Indirect Communication - Synchronization

- OS view vs Programming Languages (PL) view of synchronous communication
- OS view
  - synchronous send ⇒sender blocks until message has been copied from application buffers to kernel buffer
  - Asynchronous send ⇒sender continues processing after notifying OS of the buffer in which the message is stored; have to be careful to not overwrite buffer until it is safe to do so
- PL view:
  - synchronous send ⇒sender blocks until message has been received by the receiver
  - asynchronous send ⇒sender carries on with other tasks after sending message (OS view of synchronous communication is asynchronous from the PL viewpoint)

IPC - Message Passing - Indirect Communication

- 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**

IPC - Message Passing - Buffering

- Queue of messages attached to the link
- Implemented in one of three ways.
    - Zero capacity – 0 messages Sender must wait for receiver (rendezvous).
    - Bounded capacity – finite length of n messages or N bytes. Sender must wait if link full.
    - Unbounded Capacity - Infinite length of n messages or N bytes. Sender never waits

Inter-Process Communication (IPC) OS dependent Mechanisms

Pipes − Communication between two related processes. The mechanism is half duplex meaning the first process communicates with the second process. To achieve a full duplex i.e., for the second process to communicate with the first process another pipe is required.

FIFO − Communication between two unrelated processes. FIFO is a full duplex, meaning the first process can communicate with the second process and vice versa at the same time.

Message Queues − Communication between two or more processes with full duplex capacity. The processes will communicate with each other by posting a message and retrieving it out of the queue. Once retrieved, the message is no longer available in the queue.

Shared Memory − Communication between two or more processes is achieved through a shared piece of memory among all processes. The shared memory needs to be protected from each other by synchronizing access to all the processes.

Semaphores − Semaphores are meant for synchronizing access to multiple processes. When one process wants to access the memory (for reading or writing), it needs to be locked (or protected) and released when the access is removed. This needs to be repeated by all the processes to secure data.

Signals − Signal is a mechanism to communication between multiple processes by way of signaling. This means a source process will send a signal (recognized by number) and the destination process will handle it accordingly.

IPC - Pipes

- Pipe is a communication medium between two or more related or interrelated processes.
- It can be either within one process or a communication between the child and the parent processes.
- Communication can also be multi-level such as communication between the parent, the child and the grand-child, etc.
- Communication is achieved by one process writing into the pipe and other reading from the pipe.
- To achieve the pipe system call, create two files, one to write into the file and another to read from the file.

IPC - Pipes

- Pipe mechanism can be viewed with a real-time scenario such as filling petrol with the pipe into some vehicle, say a petrol tank, and someone retrieving it, say with a nozzle.

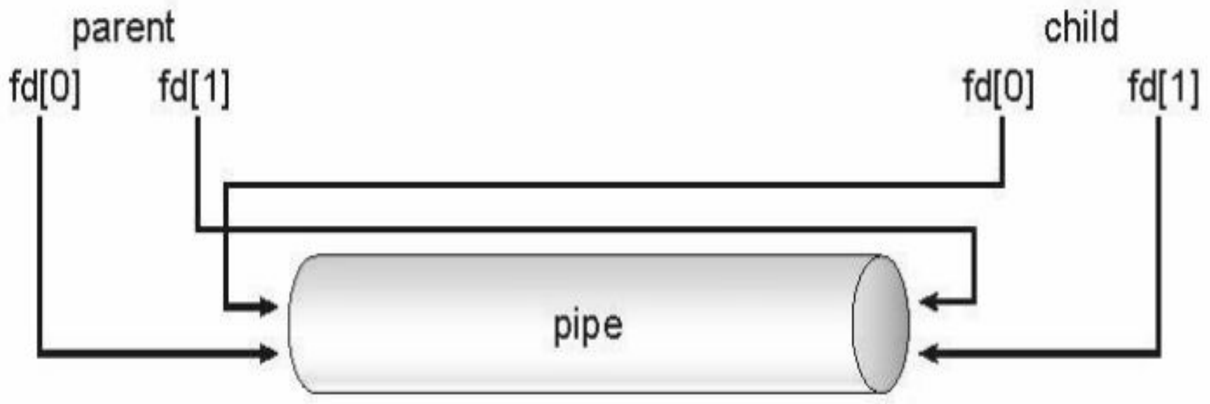
- The filling process is nothing but writing into the pipe and the reading process is nothing but retrieving from the pipe. This implies that one output (petrol) is input for the other (tank).

IPC - Pipes

- Acts as a conduit allowing two processes to communicate
- Issues:
  - Is communication unidirectional or bidirectional?
  - In the case of two-way communication, is it half or full-duplex?
  - Must there exist a relationship (i.e., ***parent-child***) between the communicating processes?
  - Can the pipes be used over a network?
- Ordinary pipes – cannot be accessed from outside the process that created it. Typically, a parent process creates a pipe and uses it to communicate with a child process that it created.
- Named pipes – can be accessed without a parent-child relationship

IPC - Pipes

- Ordinary Pipes allow communication in standard producer-consumer style
- Producer writes to one end (the **write-end** of the pipe)
- Consumer reads from the other end (the **read-end** of the pipe)
- Ordinary pipes are therefore unidirectional
- Require parent-child relationship between communicating processes



- Windows calls these **anonymous pipes**

THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

nitin.pujari@pes.edu

For Course Deliverables by the Anchor Faculty click on www.pesuacademy.com