Inter Process Communication (IPC)

By GeeksforGeeks

Published: 2017-01-24 · Archived: 2026-04-06 00:47:31 UTC

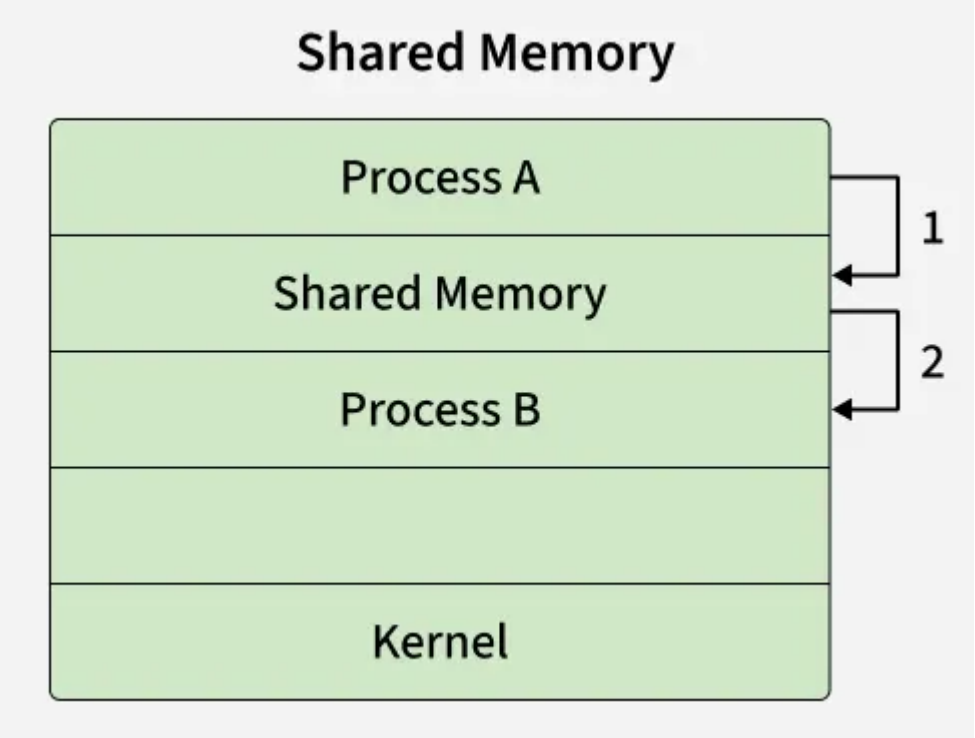
Last Updated : 29 Jan, 2026

Inter-Process Communication or IPC is a mechanism that allows processes to communicate and share data with each other while they are running. Since each process has its own memory space, IPC provides controlled methods for exchanging information and coordinating actions. It helps processes work together efficiently and safely in an operating system.

- It helps processes synchronize their activities, share information and avoid conflicts while accessing shared resources.
- There are two method of IPC, shared memory and message passing. An operating system can implement both methods of communication.

Example: A simple example of IPC is a bank ATM system, where one process reads the card and PIN, another checks the account balance, and a third dispenses cash. These processes communicate and coordinate to complete the transaction correctly.

Communication between processes using shared memory requires processes to share some variable and it completely depends on how the programmer will implement it. Processes can use shared memory for extracting information as a record from another process as well as for delivering any specific information to other processes.

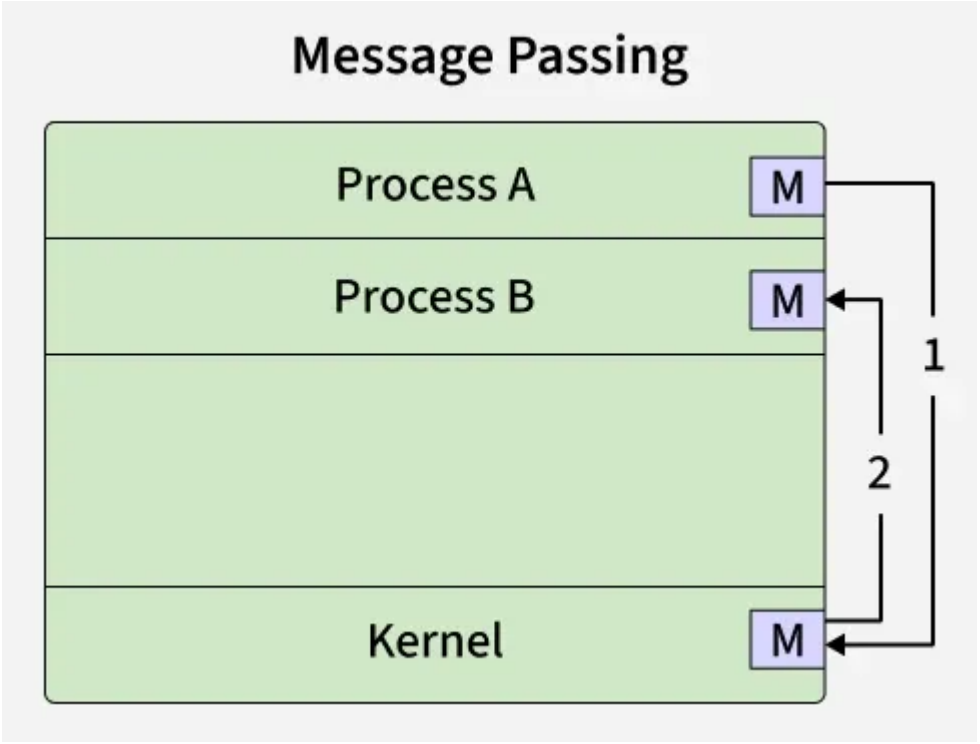


Shared Memory

- In the above shared memory model, a common memory space is allocated by the kernel.
- Process A writes data into the shared memory region (Step 1).
- Process B can then directly read this data from the same shared memory region (Step 2).
- Since both processes access the same memory segment, this method is fast but requires synchronization mechanisms (like semaphores) to avoid conflicts when multiple processes read/write simultaneously.
- Example: Multiple people can edit the document at the same time in shared google doc.

Message Passing

Message Passing is a method where processes communicate by sending and receiving messages to exchange data. One process sends a message and the other process receives it, allowing them to share information. Message Passing can be achieved through different methods like Sockets, Message Queues or Pipes.



Message Passing

- In the above message passing model, processes exchange information by sending and receiving messages through the kernel.
- Process A sends a message to the kernel (Step 1).
- The kernel then delivers the message to Process B (Step 2).
- Here, processes do not share memory directly. Instead, communication happens via system calls (send(), recv(), or similar).
- This method is simpler and safer than shared memory because there’s no risk of overwriting shared data, but it incurs more overhead due to kernel involvement.
- Example: Multiple people send updates to a group chat, but each message goes through the server before others see it, like processes sending messages instead of directly sharing memory.

> Please refer Methods in Inter process Communication for more details.

Problems in Inter-Process Communication (IPC):

Inter-Process Communication (IPC) faces challenges when multiple processes share resources. Improper synchronization can cause race conditions, deadlock, and starvation, while shared data may suffer data inconsistency. IPC also adds overhead and can raise security issues. Managing many processes can lead to scalability problems.

Some common classical IPC problem are:

Dining Philosophers Problem

This problem illustrates deadlock and starvation. The **Dining Philosophers Problem** involves five philosophers sitting around a table, each needing two forks (shared resources) to eat. If all philosophers pick up one fork at the same time, none can eat, resulting in deadlock.

Solution

- Use semaphores or monitors to control access to forks.
- Allow only one philosopher to pick forks at a time or limit eating to four philosophers.
- Enforce an order of picking forks to avoid circular wait.
- Prevents deadlock and starvation.

Producer–Consumer Problem

This problem deals with synchronization and buffer management. The **Producer–Consumer Problem** describes producers generating data and placing it in a shared buffer, while consumers remove data from it. The main challenge is preventing producers from adding data to a full buffer and consumers from removing data from an empty buffer.

Solution

- Use mutex to ensure mutual exclusion on the shared buffer.
- Use counting semaphores to track empty and full buffer slots.
- Producer waits if buffer is full; consumer waits if buffer is empty.
- Ensures proper synchronization and data consistency.

Readers–Writers Problem

This problem focuses on concurrent access to shared data. The **Readers–Writers Problem** allows multiple readers to read data simultaneously, while writers require exclusive access. The challenge is avoiding starvation of either readers or writers.

Solution

- Use reader–writer locks or semaphores.

- Allow multiple readers to read simultaneously.
- Grant writers exclusive access to shared data.
- Apply priority rules to avoid starvation.

Sleeping Barber Problem

This problem demonstrates process coordination. The **Sleeping Barber Problem** models a barber who sleeps when there are no customers and is awakened when a customer arrives and a chair is available. The difficulty lies in managing waiting chairs and customer arrivals correctly.

Solution

- Use semaphores to manage customer arrival and barber availability.
- Barber sleeps when no customers are present.
- Customers wait if chairs are available; otherwise, they leave.
- Ensures proper process coordination without deadlock.
