Inter process communication

A process can be of two type:

- Independent process.
- Co-operating process.

Processes can communicate with each other using these two ways:

1. Shared Memory
2. Message passing

Shared Data between the two Processes

Processes can use shared memory for extracting information as a record from other process as well as for delivering any specific information to other process.

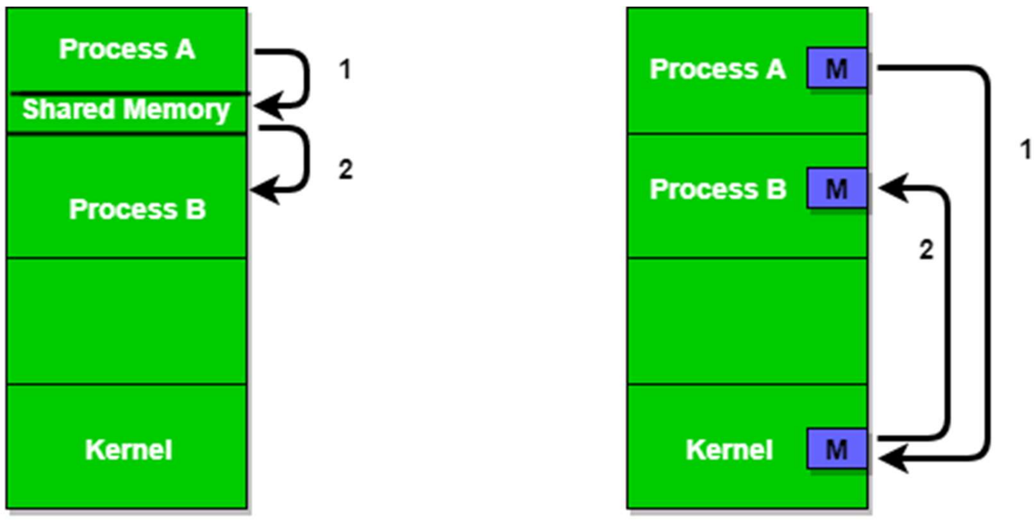


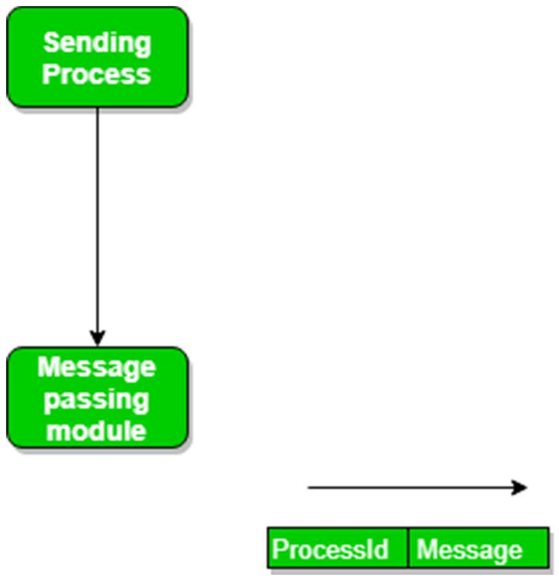
Figure 1 - Shared Memory and Message Passing

Messaging Passing Method

If two processes p1 and p2 want to communicate with each other, they proceed as follow:

- Establish a communication link (if a link already exists, no need to establish it again.)
- Start exchanging messages using basic primitives.
  We need at least two primitives:

– **send**(message, destinaion) or **send**(message)
– **receive**(message, host) or **receive**(message)



Message Passing through Communication Link.

Direct Communication links are implemented when the processes use specific process identifier for the communication but it is hard to identify the sender ahead of time.

In-directed Communication is done via a shred mailbox (port), which consists of queue of messages. Sender keeps the message in mailbox and receiver picks them up.

Message Passing through Exchanging the Messages.

Synchronous and Asynchronous Message Passing:

Blocking is considered **synchronous** and **blocking send** means the sender will be blocked until the message is received by receiver. Similarly, **blocking receive** has the receiver block until a message is available. Non-blocking is considered **asynchronous** and Non-blocking send has the sender sends the message and continue. Similarly, Non-blocking receive has the receiver receive a valid message or null.

There are basically three most preferred combinations:

- Blocking send and blocking receive
- Non-blocking send and Non-blocking receive
- Non-blocking send and Blocking receive (Mostly used)

In Direct message passing, The process which want to communicate must explicitly name the recipient or sender of communication.
e.g. **send(p1, message)** means send the message to p1.
similarly, **receive(p2, message)** means receive the message from p2.

In Indirect message passing, processes uses mailboxes (also referred to as ports) for sending and receiving messages. Each mailbox has a unique id and processes can communicate only if they share a mailbox. Link established only if processes share a common mailbox and a single link can be associated with many processes. Each pair of processes can share several communication links and these link may be unidirectional or bi-directional.

Port is an implementation of such mailbox which can have multiple sender and single receiver. It is used in client/server application (Here server is the receiver). The port is owned by the receiving process and created by OS on the request of the receiver process and can be destroyed either on request of the same receiver process or when the receiver terminates itself.

Examples of IPC systems

1. Posix : uses shared memory method.
2. Mach : uses message passing
3. Windows XP : uses message passing using local procedural calls

Communication in client/server Architecture:
There are various mechanism:

- Pipe
- Socket
- Remote Procedural calls (RPCs)

Pipes

Pipe is widely used for communication between two related processes. This is a half-duplex method, so the first process communicates with the second process. However, in order to achieve a full-duplex, another pipe is needed.

Message Passing:

It is a mechanism for a process to communicate and synchronize. Using message passing, the process communicates with each other without resorting to shared variables.

Message Queues:

A message queue is a linked list of messages stored within the kernel. It is identified by a message queue identifier. This method offers communication between single or multiple processes with full-duplex capacity.

Direct Communication:

In this type of inter-process communication process, should name each other explicitly. In this method, a link is established between one pair of communicating processes, and between each pair, only one link exists.

Indirect Communication:

Indirect communication establishes like only when processes share a common mailbox each pair of processes sharing several communication links. A link can communicate with many processes. The link may be bi-directional or unidirectional.

Shared Memory:

Shared memory is a memory shared between two or more processes that are established using shared memory between all the processes. This type of memory requires to protected from each other by synchronizing access across all the processes.

FIFO:

Communication between two unrelated processes. It is a full-duplex method, which means that the first process can communicate with the second process, and the opposite can also happen.

Pipes (Same Process) –
This allows flow of data in one direction only. Analogous to simplex systems (Keyboard). Data from the output is usually buffered until input process receives it which must have a common origin.

Names Pipes (Different Processes) –
This is a pipe with a specific name it can be used in processes that don’t have a shared common process origin. E.g. is FIFO where the details written to a pipe is first named.

Message Queuing –
This allows messages to be passed between processes using either a single queue or several message queue. This is managed by system kernel these messages are coordinated using an API.

Semaphores –
This is used in solving problems associated with synchronization and to avoid race condition. These are integer values which are greater than or equal to 0.

Shared memory –
This allows the interchange of data through a defined area of memory. Semaphore values have to be obtained before data can get access to shared memory.

Sockets –
This method is mostly used to communicate over a network between a client and a server. It allows for a standard connection which is computer and OS independent.

http://simplestcodings.blogspot.com/2010/08

Pipe program

https://www.geeksforgeeks.org/pipe-system-call/

https://tldp.org/LDP/lpg/node11.html

Socket program

https://www.programminglogic.com/example-of-client-server-program-in-c-using-sockets-and-tcp/

message queue

http://simplestcodings.blogspot.com/2010/08/ipc-message-queue-implementation-in-c.html

FIFO

https://www.geeksforgeeks.org/named-pipe-fifo-example-c-program/

shared memory

https://www.geeksforgeeks.org/ipc-shared-memory/

http://simplestcodings.blogspot.com/2010/08/ipc-shared-memory-implementation-in-c.html