Interprocess Communication Mechanism

Exchange of data between two or more separate, independent processes/threads is possible using IPC. Operating systems provide facilities/resources for Inter-Process Communications (IPC), such as message queues, semaphores and shared memory.

A complex programming environment often uses multiple cooperating processes to perform related operations. These processes must communicate with each other and share resources and information. The Kernel must provide mechanisms that make this possible. These mechanisms are collectively referred to as interprocess communication.

Distributed computing systems make use of these facilities/resources to provide Application Programming Interface (API) which allows IPC to be programmed at a higher level of abstraction, (e.g., send and receive).

Five types of inter-process communication are as follows :

1. Shared memory permits processes to communicate by simply reading and writing to a specified memory location.
2. Mapped memory is similar to shared memory, except that it is associated with a file in the file system.
3. Pipes permit sequential communication from one process to a related process.
4. FIFOs are similar to pipes, except that unrelated processes can communicate because the pipe is given a name in the file system.
5. Sockets support communication between unrelated processes even on different computers.

Name	Description	Scope	Use
File	Data is written to and read from a typical UNIX file. Any number of processes can interoperate.	Local	Sharing large data sets.

Pipe	Data is transferred between two processes using dedicated file descriptors. Communication occurs only between a parent and child process.	Local	Simple data sharing, such as producer and consumer.
Named pipe	Data is exchanged between processes via dedicated file descriptors. Communication can occur between any two peer processes on the same host.	Local	Producer and consumer or command-and-contro 1, as demonstrated with MySQL server and its command-line query utility.
Signal	An interrupt alerts the application to a specific condition.	Local	Cannot transfer data in a signal, so mostly useful for process management.
Shared memory	Information is shared by reading and writing from a common segment of memory.	Local	Cooperative work of any kind, especially if security is required.
Socket	After special setup, data is transferred using common input/output operations.	Local or remote	Network services such as FTP, ssh, and the Apache Web Server.

Purposes of IPC

Data transfer : One process may wish to send data to another process.

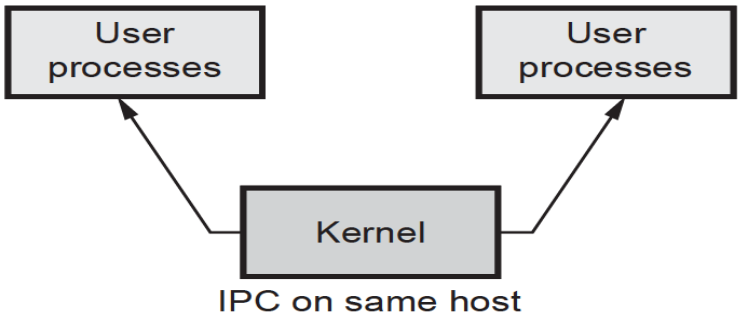
Sharing data : Multiple processes may wish to operate on shared data, such that if a process modifies the data, that change will be immediately visible to other processes sharing it

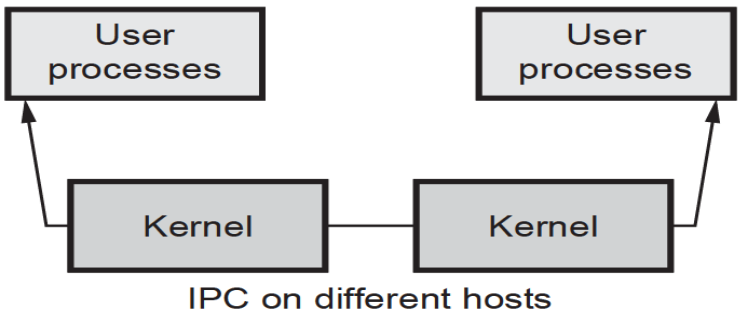
Event modification : A process may wish to notify another process or set of processes that some event has occurred.

Resource sharing : The Kernel provides default semantics for resource allocation; they are not suitable for all application.

Process control : A process such as a debugger may wish to assume complete control over the execution of another process.

IPC has two forms : IPC on same host and IPC on different hosts.





IPC on different hosts

IPC is used for 2 functions :

1. **Synchronization :** Used to coordinate access to resources among processes and also to coordinate the execution of these processes. They are record locking, Semaphores, mutexes and condition variables.
2. **Message passing :** Used when processes wish to exchange information. Message passing takes several forms such as : Pipes, FIFOs, Message queues and Shared memory.

Features of Message Passing

1. **Simplicity :** Message passing system should be simple and easy to use. It should be possible to communicate with old and new applications.
2. **Uniform semantics :** Message passing is used for two types of IPC.
    1. **Local communication :** Communicating processes are on the same node.
    2. **Remote communication :** Communicating processes are on the different nodes.

3. **Efficiency :** IPC become so expansive if message passing system is not effective. Users try avoiding to IPC for their applications. Message passing system will become more efficient if we try to avoid more message exchanges during communication process. For examples :

a. Avoiding the costs of establishing and terminating connection.
b. Minimizing the costs of maintaining the connections.
c. Piggybacking of acknowledgement.

4. **Reliability :** Distributed systems are prone to different catastrophic events such as node crashes or physical link failures. Loss of message because of communication link fails. To handle the loss messages, we required acknowledgement and retransmission policy. Duplicate message is one of the major problems. This happens because of timeouts or events of failures.
5. **Correctness :** Correctness is a feature related to IPC protocols for group communication. Issues related to correctness are as follows :

1. **Atomicity :** Every message sent to a group of receivers will be delivered to either all of them or none of them.
2. **Ordered delivery :** Messages arrive to all receivers in an order acceptable to the application.
3. **Survivability :** Messages will be correctly delivered despite partial failures of processes, machine, or communication links.
6. **Security :** Message passing system must provide a secure end to end communication.
7. **Portability :** Message passing system should itself be portable.

IPC Message Format

Message passing system requires the synchronization and communication between the two processes. Message passing used as a method of communication in microkernels. Message passing systems come in many forms. Messages sent by a process can be either fixed or variable size. The actual function of message passing is normally provided in the form of a pair of primitives.

a) Send (destination_name, message)

b) Receive (source_name, message)

Send primitive is used for sending a message to destination. Process sends information in the form of a message to another process designated by a destination. A process receives information by executing the receive primitive, which indicates the source of the sending process and the message.

Design characteristics of message system for IPC.

1. Synchronization between the process
2. Addressing
3. Format of the message
4. Queueing discipline

Issues in IPC by Message Passing

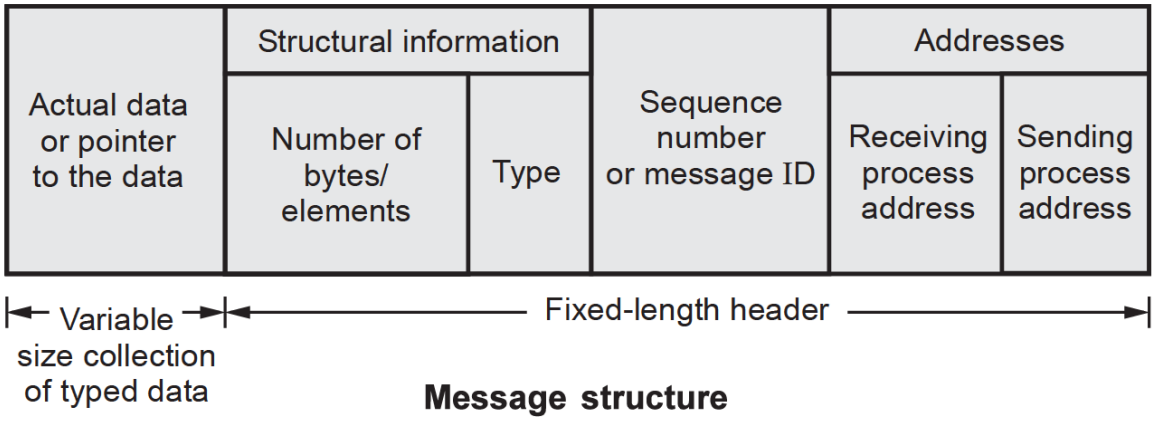
Message is a block of information.

A message is a meaningful formatted block of information sent by the sender process to the receiver process.

The message block consists of a fixed length header followed by a variable size collection of typed data objects.

The header block of a message may have the following elements :

1. **Address :** A set of characters that uniquely identify both the sender and receiver.
2. **Sequence number :** It is the message identifier to identify duplicate and lost messages in case of system failures.
3. **Structural information :** It has two parts. The type part that specifies whether the data to be sent to the receiver is included within the message or the message only contains a pointer to the data. The second part specifies length of the variable-size message.



Message structure

Some important issues to be considered for the design of an IPC protocol based message passing system :

I. The sender's identity
II. The receiver's identity
III. Number of receivers
IV. Guaranteed acceptance of sent messages by the receiver
V. Acknowledgement by the sender
VI. Handling system crashes or link failures
VII. Handling of buffers
VIII. Order of delivery of messages

IPC Synchronization

Send operation can be synchronous or asynchronous. Receive operation can be blocking or nonblocking.

Sender and receiver process can be blocking mode or nonblocking mode. Different possibility of sender and receivers are as follows :

1. Blocking send, blocking receive
2. Nonblocking send, block receive
3. Nonblocking send, Nonblocking receive

Blocking send, blocking receive

Blocking send must wait for the receiver to receive the message, synchronous communication is an example of blocking send. Both processes (sender and receiver) are blocked untill the message is delivered.

Rendezvous : Sending a message to another process leaving the sender process suspended until the message is received and processed.

Nonblocking send, blocking receive

Sender is free to send the messages but receiver is blocked until the requested message arrives.

Asynchronous communication is an example of nonblocking send. Asynchronous communication with nonblocking sends increases throughput by reducing the time that processes spend waiting.

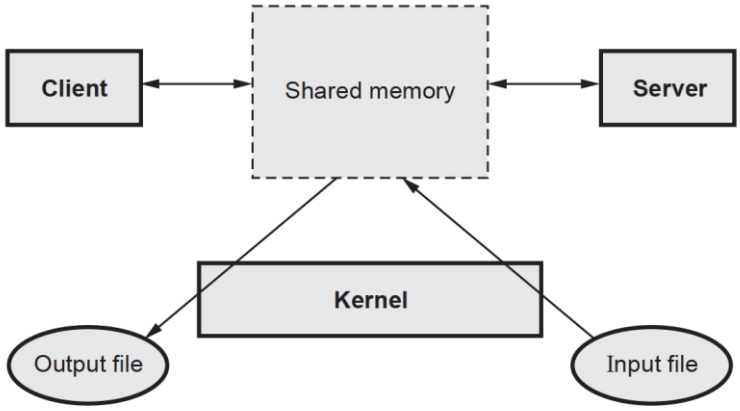
Natural concurrent programming task uses the nonblocking send. Nonblocking send is the overhead on the programmer for determine that a message has been received or not.

Shared Memory

A region of memory that is shared by co-operating processes is established. Processes can then exchange information by reading and writing data to the shared region.

Shared memory allows maximum speed and convenience of communication, as it can be done at memory speeds when within a computer. Shared memory is faster than message passing, as message-passing systems are typically implemented using system calls and thus require the more time-consuming task of Kernel intervention.

In contrast, in shared-memory systems, system calls are required only to establish shared-memory regions.



Client / server with shared memory

Advantages

1. Good for sharing large amount of data. 2. Very fast.

Limitations

1. No synchronization provided - applications must create their own.
2. Alternative to shared memory is mmap system call, which maps file into the address space of the caller.