Inter-process communication :

**Shared memory :**

[Inter Process Communication](https://www.geeksforgeeks.org/inter-process-communication/) through shared memory is a concept where two or more process can access the common memory. And communication is done via this shared memory where changes made by one process can be viewed by another process.

Shared memory allows to have many process interacting with many processes and the exchanges are asynchronized, meaning that anyone can access the data from the shared memory at any time.

Pipe:

Runs two or more commands in order to do complex tasks.

The Pipe is a command in Linux that lets you use two or more commands such that output of one command serves as input to the next. In short, the output of each process directly as input to the next one like a pipeline. The symbol '|' denotes a pipe.

Pipes help you mash-up two or more commands at the same time and run them consecutively. You can use powerful commands which can perform complex tasks in a jiffy.

Pipes in c :

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.

Pipe mechanism can be viewed with a real-time scenario such as filling water with the pipe into some container, say a bucket, and someone retrieving it, say with a mug. 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 (water) is input for the other (bucket).

Mkfifo:

In computing, a named pipe (also known as a **FIFO**) is one of the methods for intern-process communication.

* It is an extension to the traditional pipe concept on Unix. A traditional pipe is “unnamed” and lasts only as long as the process.
* A named pipe, however, can last as long as the system is up, beyond the life of the process. It can be deleted if no longer used.
* Usually a named pipe appears as a file and generally processes attach to it for inter-process communication. A FIFO file is a special kind of file on the local storage which allows two or more processes to communicate with each other by reading/writing to/from this file.
* A FIFO special file is entered into the filesystem by calling *mkfifo()* in C. Once we have created a FIFO special file in this way, any process can open it for reading or writing, in the same way as an ordinary file. However, it has to be open at both ends simultaneously before you can proceed to do any input or output operations on it.

**Example Programs to illustrate the named pipe:**There are two programs that use the same FIFO. Program 1 writes first, then reads. The program 2 reads first, then writes. They both keep doing it until terminated.

Socket :

A **Unix domain socket** or **IPC socket** ([inter-process communication](https://en.wikipedia.org/wiki/Inter-process_communication) socket) is a data [communications endpoint](https://en.wikipedia.org/wiki/Communication_endpoint) for exchanging data between processes executing on the same host operating system. Valid socket types in the UNIX domain are:[[1]](https://en.wikipedia.org/wiki/Unix_domain_socket#cite_note-man-unix-sockets-1)

* SOCK\_STREAM (compare to [TCP](https://en.wikipedia.org/wiki/Transmission_Control_Protocol)) – for a stream-oriented socket
* SOCK\_DGRAM (compare to [UDP](https://en.wikipedia.org/wiki/User_Datagram_Protocol)) – for a datagram-oriented socket that preserves message boundaries (as on most UNIX implementations, UNIX domain datagram sockets are always reliable and don't reorder datagrams)
* SOCK\_SEQPACKET (compare to [SCTP](https://en.wikipedia.org/wiki/SCTP)) – for a sequenced-packet socket that is connection-oriented, preserves message boundaries, and delivers messages in the order that they were sent

The Unix domain socket facility is a standard component of [POSIX](https://en.wikipedia.org/wiki/POSIX) [operating systems](https://en.wikipedia.org/wiki/Operating_system).

The [API](https://en.wikipedia.org/wiki/API) for Unix domain sockets is similar to that of an [Internet socket](https://en.wikipedia.org/wiki/Internet_socket), but rather than using an underlying network protocol, all communication occurs entirely within the operating system [kernel](https://en.wikipedia.org/wiki/Kernel_(operating_system)). Unix domain sockets may use the file system as their address [name space](https://en.wikipedia.org/wiki/Name_space). (Some operating systems, like Linux, offer additional namespaces.) Processes reference Unix domain sockets as file system [inodes](https://en.wikipedia.org/wiki/Inode), so two processes can communicate by opening the same socket.

In addition to sending data, processes may send [file descriptors](https://en.wikipedia.org/wiki/File_descriptor) across a Unix domain socket connection using the sendmsg() and recvmsg() system calls. This allows the sending processes to grant the receiving process access to a file descriptor for which the receiving process otherwise does not have access.[[2]](https://en.wikipedia.org/wiki/Unix_domain_socket#cite_note-neohapsis-2)[[3]](https://en.wikipedia.org/wiki/Unix_domain_socket#cite_note-linux-cmsg-man-page-3) This can be used to implement a rudimentary form of [capability-based security](https://en.wikipedia.org/wiki/Capability-based_security).[[4]](https://en.wikipedia.org/wiki/Unix_domain_socket#cite_note-wheeler-secure-linux-howto-4) For example, this allows the [Clam AntiVirus](https://en.wikipedia.org/wiki/Clam_AntiVirus) scanner to run as an [unprivileged](https://en.wikipedia.org/wiki/Privilege_(computing)#Unix) [daemon](https://en.wikipedia.org/wiki/Daemon_(computing)) on Linux and BSD, yet still read any file sent to the daemon's Unix domain socket.

File descriptor :

In [Unix](https://en.wikipedia.org/wiki/Unix) and [related](https://en.wikipedia.org/wiki/Unix-like) computer operating systems, a **file descriptor** (**FD**, less frequently **fildes**) is an abstract indicator ([handle](https://en.wikipedia.org/wiki/Handle_(computing))) used to access a [file](https://en.wikipedia.org/wiki/File_(computing)) or other [input/output](https://en.wikipedia.org/wiki/Input/output) [resource](https://en.wikipedia.org/wiki/System_resource), such as a [pipe](https://en.wikipedia.org/wiki/Pipe_(Unix)) or [network socket](https://en.wikipedia.org/wiki/Network_socket). File descriptors form part of the [POSIX](https://en.wikipedia.org/wiki/POSIX) [application programming interface](https://en.wikipedia.org/wiki/Application_programming_interface). A file descriptor is a non-negative [integer](https://en.wikipedia.org/wiki/Integer), generally represented in the [C](https://en.wikipedia.org/wiki/C_(programming_language)) programming language as the type [int](https://en.wikipedia.org/wiki/C_data_types#Basic_types) (negative values being reserved to indicate "no value" or an error condition).