# MCS 172 - UNIX OPERATING SYSTEM

23-11-2020

**Assignment**

Inter-Process Communication

Rajkumar B L : 2047120

## Inter-Process Communication

IPC is a developed mechanism to allow multiple processes to communicate with each other and synchronize their action. The data exchanged during the communication passes through the kernel except for the shared memory IPC technique. The three Inter-Process Communication IPC techniques discussed in this article are Pipes, FIFO, and Message-queue.

### Pipe:

The pipe is one of the oldest IPC techniques used in UNIX. It is a communication medium that carries a byte stream between two related or inter-related processes. The data flow here is unidirectional and requires the use of two pipes to achieve two-way communication. Upon completion of process execution, the channel will get closed.

```
System call: int pipe(int pipedes[2]);
```

The above system call creates a pipe for one way communication, where descriptor pipedes[0] is used for reading and pipedes[1] for writing.

Pros: -

- Simple Technique of Inter-Process Communication
- Best used for inter-related (Parent-child) processes

Cons: -

- Cannot be used for broadcasting
- Only for inter-related processes
- Data in a pipe is a byte stream with no structure.
- Is a memory buffer, data not stored.
- A pipe is local to the system, and we cannot use it for communication across networks

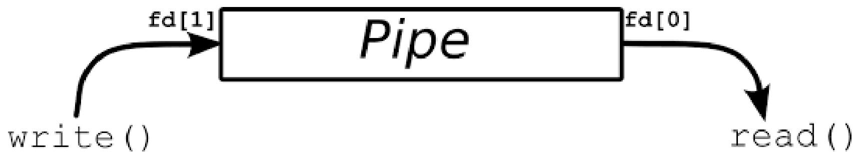


*Figure 1: https://www.cse.iitd.ernet.in/~sbansal/os/previous_years/2011/pa1/beej%27s%20guide%20to%20unix%20ipc.pdf*

## FIFO:

FIFO(First In First Out) is similar to a pipe; the named pipe is also called FIFO. FIFO has a name within the UNIX file system; thus, we can access it like a regular file. The crucial difference is that we can use FIFO to communicate between two unrelated processes, and a single named pipe (FIFO) can achieve two-way communication. It is a half-duplex communication as one process writes and the other reads. FIFO exists even after completion of process execution, unlike pipes, and requires an explicit command to delete/unlink.

```
System call: int mkfifo(const char *path, mode_t mode);
```

The above system call creates a named pipe as specified by the path, and the mode parameter sets the permissions of the file. After this, we can read and write to FIFO just like a regular file in the system. Where blocking is automatically implemented upon execution reading/writing operation to avoid the deadlock.

Pros: -

- Allows IPC for unrelated processes
- Bi-directional, we can use the same FIFO for reading/writing
- No synchronization mechanism is needed
- Multiple process communication is possible
- Ease of use, like a regular file

Cons: -

- Less secure than a pipe, as any process with valid permission can access them
- Deletion of FIFO by one process can affect other processes that depend on it
- No methodology to direct the data to a particular process, in case of multiple receivers
- Can be created only in the local file system of the host and not in the NFS file system

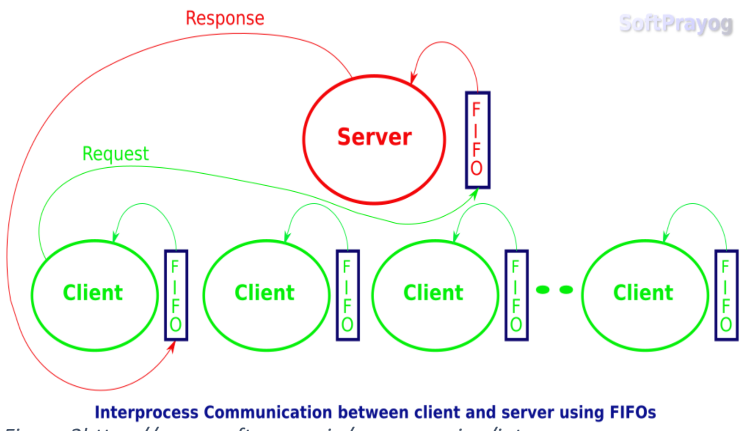


*Figure 2https://www.softprayog.in/programming/interprocess-communication-using-fifos-in-linux*

- Careful programming is required to avoid deadlock

## Shared Memory:

Shared Memory is one of the best and simple methods used in inter-process communication, where two or more processes share a specific chunk of memory to communicate amongst them. Predominantly, unrelated processes use shared memory/message queue for their communication purpose. First, the shared memory segment (SHM) is created and connected through the shmget() call. After creating, attach the process to the shared memory segment using the shmat() call. The attaching process must have proper permissions for the attachment system call. When attached, the process can read or write to the SHM. Upon completion, detach the process from the shared memory segment using shmdt() call. Control operations on SHM with shmctl() call.

```
Create: int shmget(key_t key, size_t size, int shmflg);
Attach: void *shmat(int shmid, void *shmaddr, int shmflg);
Detach: int shmdt(void *shmaddr);
Control: shmctl(shmid, IPC_RMID, NULL); //Deletes shm
```

Pros: -

- Fast bi-directional communication among any number o processes
- Does not require extra kernel buffer, saves resources
- Suitable for sharing the extensive amount of data

Cons: -

- No synchronization provided
- The race condition is possible, so locking is required
- Lack of data protection from OS

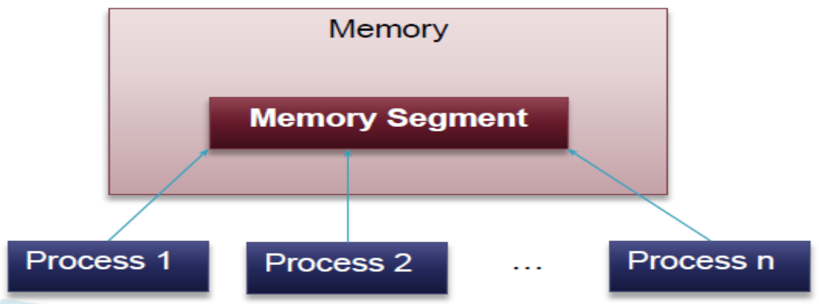
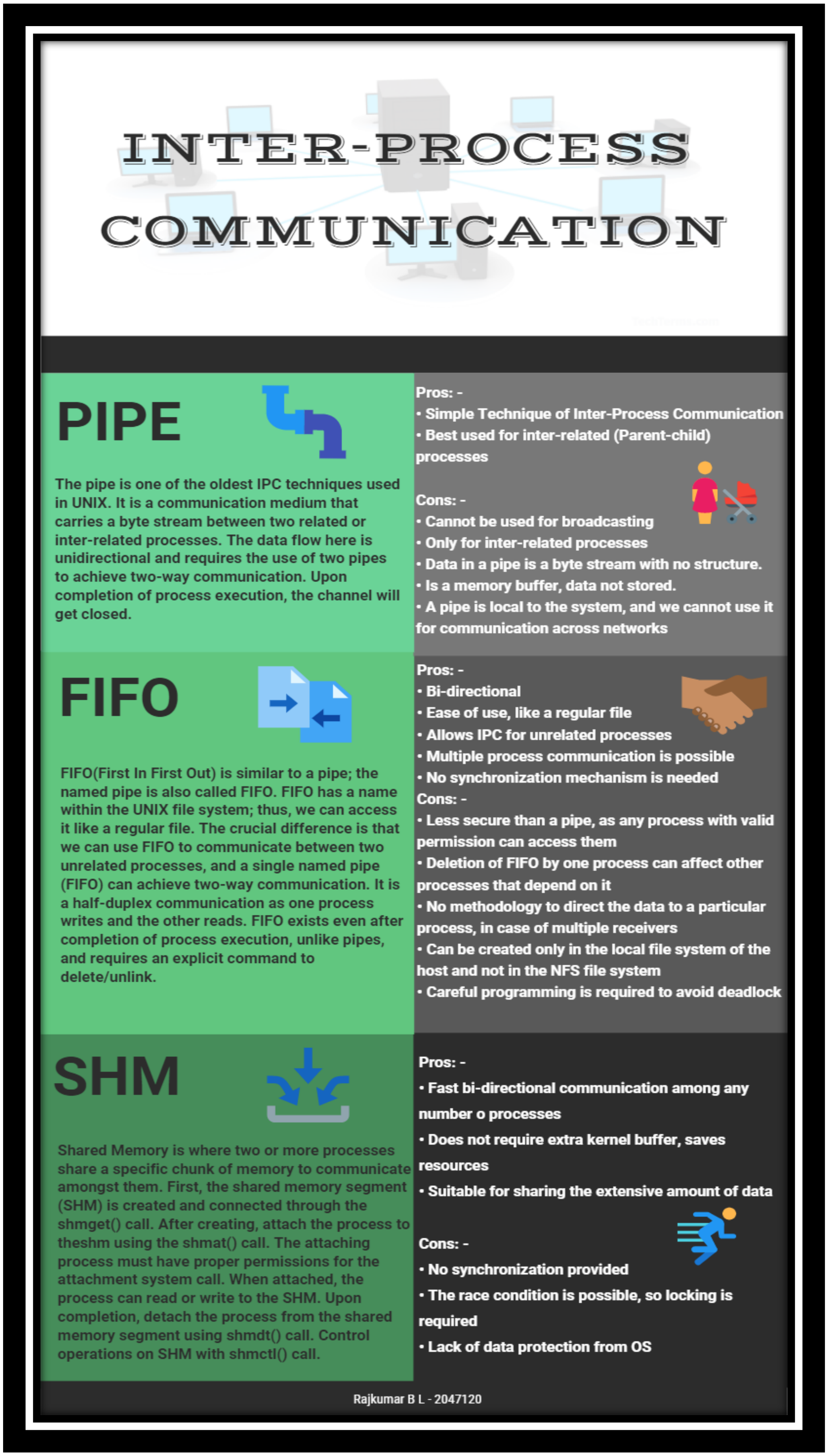


*Figure 3: Lecture Slide*

## Infographic Representation:



INTER-PROCESS COMMUNICATION

**PIPE**

The pipe is one of the oldest IPC techniques used in UNIX. It is a communication medium that carries a byte stream between two related or inter-related processes. The data flow here is unidirectional and requires the use of two pipes to achieve two-way communication. Upon completion of process execution, the channel will get closed.

Pros: -
- Simple Technique of Inter-Process Communication
- Best used for inter-related (Parent-child) processes

Cons: -
- Cannot be used for broadcasting
- Only for inter-related processes
- Data in a pipe is a byte stream with no structure.
- Is a memory buffer, data not stored.
- A pipe is local to the system, and we cannot use it for communication across networks

**FIFO**

FIFO(First In First Out) is similar to a pipe; the named pipe is also called FIFO. FIFO has a name within the UNIX file system; thus, we can access it like a regular file. The crucial difference is that we can use FIFO to communicate between two unrelated processes, and a single named pipe (FIFO) can achieve two-way communication. It is a half-duplex communication as one process writes and the other reads. FIFO exists even after completion of process execution, unlike pipes, and requires an explicit command to delete/unlink.

Pros: -
- Bi-directional
- Ease of use, like a regular file
- Allows IPC for unrelated processes
- Multiple process communication is possible
- No synchronization mechanism is needed

Cons: -
- Less secure than a pipe, as any process with valid permission can access them
- Deletion of FIFO by one process can affect other processes that depend on it
- No methodology to direct the data to a particular process, in case of multiple receivers
- Can be created only in the local file system of the host and not in the NFS file system
- Careful programming is required to avoid deadlock

**SHM**

Shared Memory is where two or more processes share a specific chunk of memory to communicate amongst them. First, the shared memory segment (SHM) is created and connected through the shmget() call. After creating, attach the process to theshm using the shmat() call. The attaching process must have proper permissions for the attachment system call. When attached, the process can read or write to the SHM. Upon completion, detach the process from the shared memory segment using shmdt() call. Control operations on SHM with shmctl() call.

Pros: -
- Fast bi-directional communication among any number o processes
- Does not require extra kernel buffer, saves resources
- Suitable for sharing the extensive amount of data

Cons: -
- No synchronization provided
- The race condition is possible, so locking is required
- Lack of data protection from OS

Rajkumar B L - 2047120