MIT-WPU

।। विश्वशान्तिर्ध्रुवं ध्रुवा ।।

Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

Operating Systems

School of Computer Engineering and technology

Inter-process communication (IPC)

There are several mechanisms for *Inter-Process Communication* (IPC)

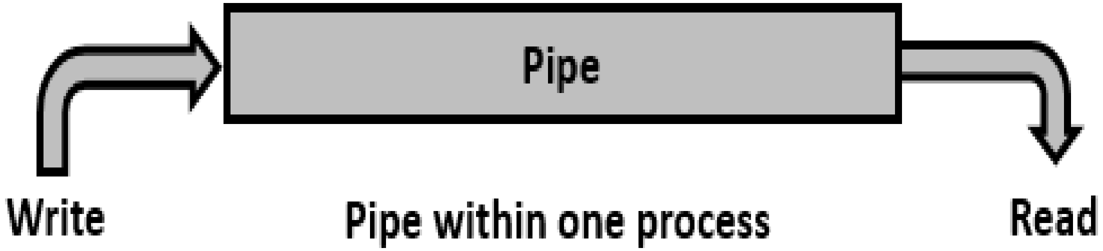
- Signals
- FIFOS (named pipes)
- Pipes
- Sockets
- Message passing
- Shared memory
- Semaphores

Pipe

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.

UNIX deals with pipes the same way it deals with files.

A process can send data **‘down’ a pipe using a write system call** and another process can receive the **data by using read at the other end.**



Programming with pipes

- Within programs a pipe is created using a system call named **pipe.**
- This system call would create a pipe for one-way communication.
- This call would return zero on success and -1 in case of failure.
- If successful, this call returns two files descriptors:

Usage

```
#include <unistd.h>
int pipe(int filedes[2]);
```

Programming with pipes

- Usage

```
#include <unistd.h>
int pipe(int filedes[2]);
```

- filesdes is a two-integer array that will hold the file descriptors that will identify the pipe If successful,
- filedes[0] will be open for reading from the pipe and
- filedes[1] will be open for writing down it.
- pipe can fail (returns -1) if it cannot obtain the file descriptors (exceeds user-limit or kernel-limit).

Programming with pipes

Here's an extremely important point: a read from a pipe only gives end-of-file if *all* file descriptors for the write end of the pipe have been closed.

Thus, after a fork, whichever process is intending to do the reading (and thus not the writing) had best close the write end of the pipe.

```
#include<unistd.h>
close(filedes)
```

The above system call closing already opened file descriptor. This implies the file is no longer in use and resources associated can be reused by any other process.

Programming with pipes

write(filedes[1], string, MAX);

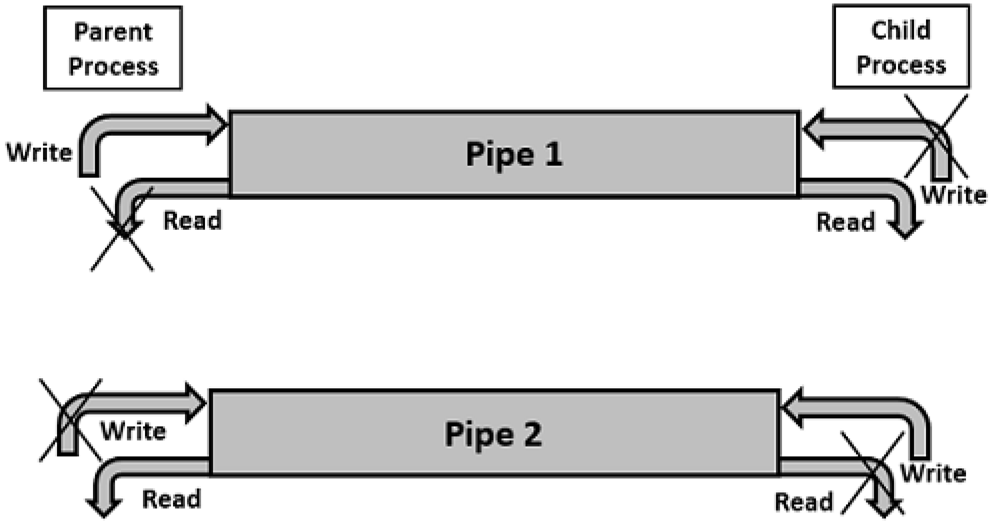
- The above system call is to write to the specified file with arguments of the file descriptor fd, string and the size of buffer.
- The file descriptor id is to identify the respective file, which is returned after calling pipe() system call.
- The file needs to be opened before writing to the file. It automatically opens in case of calling pipe() system call.
- This call would return the number of bytes written (or zero in case nothing is written) on success and -1 in case of failure. Proper error number is set in case of failure.

Programming with pipes

read(filedes[0], line, MAX);

- The above system call is to read from the specified file with arguments of file descriptor fd, string and the size of buffer.

Two-way Communication Using Pipes



- Pipe communication is viewed as only one-way communication i.e., either the parent process writes and the child process reads or vice-versa but not both.
- However, what if both the parent and the child needs to write and read from the pipes simultaneously, the solution is a two-way communication using pipes. Two pipes are required to establish two-way communication.