Pipes

pipe() function is used for inter-process communication in Linux.A pipe provides a one way flow of data. A pipe is created by pipe System Call,
The syntax of the pipe() function is:

int pipe(int pipefd[2]);

#include<unistd.h>

int pipe(int pipefd[2]);

pipe() function creates a unidirectional pipe for IPC. On success it return two file descriptors pipefd[0] and pipefd[1]. pipefd[0] is the reading end of the pipe. So, the process which will receive the data should use this file descriptor. pipefd[1] is the writing end of the pipe. So, the process that wants to send the data should use this file descriptor.

The program below creates a child process. The parent process will establish a pipe and will send the data to the child using writing end of the pipe and the child will receive that data and print on the screen using the reading end of the pipe.

How it works?

The parent process create a pipe using pipe(fd) call and then creates a child process using fork().

Then the parent sends the data by writing to the writing end of the pipe by using the fd[1] file descriptor. The child then reads this using the fd[0] file descriptor

and stores it in buffer. Then the child prints the received data from the buffer onto the screen.

IPC USING MESSAGE QUEUE

A message queue is a linked list of messages stored within the kernel and identified by a message queue identifier. A new queue is created or an existing queue opened by msgget().

New messages are added to the end of a queue by msgsnd(). Every message has a positive long integer type field, a non-negative length, and the actual data bytes (corresponding to the length), all of which are specified to msgsnd() when the message is added to a queue.

Messages are fetched from a queue by msgrcv(). We don't have to fetch the messages in a first-in, first-out order. Instead, we can fetch messages based on their type field.

All processes can exchange information through access to a common system message queue. The sending process places a message (via some (OS) message-passing module) onto a queue which can be read by another process.

Each message is given an identification or type so that processes can select the appropriate message. Process must share a common key in order to gain access to the queue in the first place.

ftok(): is use to generate a unique key.

msgget(): either returns the message queue identifier for a newly created message queue or returns the identifiers for a queue which exists with the same key value.

msgsnd(): Data is placed on to a message queue by calling msgsnd().

msgrcv(): messages are retrieved from a queue.

msgctl(): It performs various operations on a queue. Generally it is use to destroy message queue.

IPC using shared memory

Shared Memory is the fastest inter-process communication (IPC) method. The operating system maps a memory segment in the address space of several processes so that those processes can read and write in that memory segment.The overview is as shown below:
Two functions:shmget() and shmat() are used for IPC using shared memory. shmget() function is used to create the shared memory segment while shmat() function is used to attach the shared segment with the address space of the process.

Syntax (shmget()):

#include <sys/ipc.h>

#include <sys/shm.h>

int shmget(key_t key, size_t size, int shmflg);

The first parameter specifies the unique number (called key) identifying the shared segment. The second parameter is the size of the shared segment e.g.

1024 bytes or 2048 bytes. The third parameter specifies the permissions on the shared segment. On success the shmget() function returns a valid identifier while on failure it return -1.

Syntax (shmat()):

```
#include <sys/types.h>
#include <sys/shm.h>
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

shmat() is used to attach the created shared segment with the address space of the calling process. The first parameter here is the identifier which shmget() function returns on success. The second parameter is the address where to attach it to the calling process.

A NULL value of second parameter means that the system will automatically choose a suitable address. The third parameter is '0' if the second parameter is NULL, otherwise, the value is specified by SHM_RND.

How it works?

shmget() function creates a segment with key 2345, size 1024 bytes and read and write permissions for all users. It returns the identifier of the segment which gets store in shmid. This identifier is used in shmat() to attach the shared segment to the address space of the process.

NULL in shmat() means that the OS will itself attach the shared segment at a suitable address of this process.

Then some data is read from the user using read() system call and it is finally written to the shared segment using strcpy() function.