Name : Ayush. D. Chavgule
Roll no : 06 TY-CSE D1
PRN : 1032231720.

OS

Assignment - 05

Done 26/5/25

* Problem Statement :
Write a C program to implement interprocess communication using pipes.

* Objective :
- To understand the concept of pipe
- To ~~att~~ use read/write file description.
- To understand the inter-process mechanism in parent and Child process.

* Theory :

- pipes : A undirectional communication channel between proceess
- file description : enter referunces (0 - stdin, 1 - stdout, 2 - stderr, others for files/pipes).
- pipe system call : int pipe (int fd[2])
  → fd[0] for reading
  → fd[1] for writing
- msg passing : Data sent via pipe using write() and recieved using read().
- Shared memory : Another IPC mechanism (faster but more complex).

* Algorithm:

1. Create a pipe using pipe()
2. use fork() to create.
3. parent → write() a msg into pipe.
4. child → read() msg from pipe
5. Child prints msg
6. Parent waits using wait()
7. exit both process.

* Input: msg passed by parent [ex: Hello world!]

* Output: msg recieved by child [Printed: Hello world

* Conclusion: We succesfully demonstrated IPC using pipe mechanism.

* FAQ's:

1] Independent vs cooperating process.

→ * Independent process → • Does not affect or is affected to other processes.
• No data sharing, only executes its code
• ex: text editor.

* Cooperating proces → • Work with other process by sharing Data/resources.
• Requires IPC to communication.
• ex: web browser

2]. Two IPC models.

→ i) msg passing (send / recieve system calls)
   → best for distributed system
   → ex: pipes, msg Queues

ii) shared memory model
   (common region in RAM).
   → ex: shared memory segments in OS.

3].

→ pipe 1 : parent → child
pipe 2 : child → parent.

∴ Requires two pipes [one for parent another for child].

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

int main()
{
   int fd[2];
   pid_t pid;
   char write_msg[50], read_msg[50];


   if (pipe(fd) == -1)
   {
      perror("Pipe failed");
      exit(1);
   }

   pid = fork();

   if (pid < 0)
   {
      perror("Fork failed");
      exit(1);
   }

   if (pid > 0)
   {

      close(fd[0]);

     printf("Enter a message for the child: ");
     fgets(write_msg, sizeof(write_msg), stdin);


      write(fd[1], write_msg, strlen(write_msg) + 1);
      close(fd[1]);
   }
   else
   {
      close(fd[1]);
```

```
        read(fd[0], read_msg, sizeof(read_msg));
        printf("Child received: %s\n", read_msg);

        close(fd[0]);
    }
    return 0;
}
```

OUTPUT: