**Experiment 10: Demonstrate inter-process communication using pipes and signals in Linux**

## ****Aim****:

To demonstrate **inter-process communication (IPC)** using:

1. **Pipes** – to send data from one process to another.
2. **Signals** – to notify or interrupt a process based on events.

## ****Detailed Description****:

**Inter-Process Communication (IPC)** in Linux allows multiple processes to share data and coordinate actions.

### ****Pipes****:

* A **pipe** is a unidirectional communication channel.
* It allows data to flow from one process (writer) to another (reader).
* Typically used between parent-child processes.

### ****Signals****:

* **Signals** are asynchronous notifications sent to a process to notify it of an event (e.g., SIGINT, SIGUSR1).
* A process can handle signals using **signal handlers**.

In this demo:

* A **parent process** creates a **child process**.
* The child **sends data to the parent** via a **pipe**.
* The parent **sends a signal** to the child on data receipt.

## ****Execution Process****:

1. Create a **pipe** using pipe().
2. **Fork** a child process using fork().
3. In the **child**:
   * Close read end.
   * Write message into the pipe.
   * Wait for a signal from parent using pause().
4. In the **parent**:
   * Close write end.
   * Read message from pipe.
   * Send signal (SIGUSR1) to child using kill().

## ****C Program: IPC Using Pipes and Signals****

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <signal.h>

#include <sys/wait.h>

#define MSG\_SIZE 100

void signal\_handler(int sig) {

printf("Child received signal: SIGUSR1\n");

}

int main() {

int fd[2];

pid\_t pid;

char buffer[MSG\_SIZE];

if (pipe(fd) == -1) {

perror("pipe");

exit(EXIT\_FAILURE);

}

pid = fork();

if (pid < 0) {

perror("fork");

exit(EXIT\_FAILURE);

}

if (pid == 0) {

// Child Process

signal(SIGUSR1, signal\_handler);

close(fd[0]); // Close read end

const char \*message = "Hello from child";

write(fd[1], message, strlen(message) + 1);

close(fd[1]);

printf("Child process started. PID: %d\n", getpid());

pause(); // Wait for signal from parent

exit(0);

} else {

// Parent Process

close(fd[1]); // Close write end

printf("Parent: Reading from pipe...\n");

read(fd[0], buffer, sizeof(buffer));

close(fd[0]);

printf("Parent received: %s\n", buffer);

printf("Parent: Sending SIGUSR1 to child...\n");

kill(pid, SIGUSR1);

wait(NULL); // Wait for child to finish

}

return 0;

}

## ****How to Compile and Run****

1. Save as ipc\_demo.c
2. Compile:

$cc ipc\_demo.c -o ipc\_demo

1. Run:

./ipc\_demo

## ****Sample Input (No user input)****

This is an internal demonstration; the child sends a message (e.g., "Hello from child"), and the parent sends SIGUSR1.

## ****Sample Output****

Child process started. PID: 23456

Parent: Reading from pipe...

Parent received: Hello from child

Parent: Sending SIGUSR1 to child...

Child received signal: SIGUSR1