DW- Signals

Exercise 1. Inter-Process Communication using ***Signals***

Write a program in which a process creates a child process and then waits for it to finish execution. The child process displays its information (`PID`) on the screen and sends the signal `SIGUSR1` to its parent. Upon receiving this signal, the parent process will display the message "Signal received successfully" on the screen.

Exercise 2. Process Synchronization using ***Signals*** for Alternating output

Write a program that creates a child process. The parent process should alternately print "ping" to the screen, while the child process prints "pong." Use the signals `SIGUSR1` and `SIGUSR2` to facilitate communication between the parent and child. Additionally, ensure proper handling of process termination and synchronization between the two processes.

Exercise 3. Process Synchronization using ***Signals***

Write a C program that creates three child processes:

- The first child prints integers from `1` to `30`.
- The second child prints integers from `31` to `60`.
- The third child prints integers from `61` to `90`.

The output must be ordered from `1` to `90`. Propose a solution using signals (`SIGUSR1` and `SIGUSR2`) to ensure this ordered execution.

Exercise 4. Process Synchronization using ***Signals***

In this exercise, we will implement synchronization between a parent process and a child process. The execution of each process is divided into two distinct parts:

Part 1 :

- The **parent process** executes a loop for a specified number of iterations (e.g., 15 iterations).
- Simultaneously, the **child process** executes its own loop for a different number of iterations (e.g., 50 iterations).
- Each process runs its **Part 1** independently.

Part 2 :

- The **parent process** can only begin its **Part 2** after the child process has completed **Part 1**.
- Similarly, the **child process** can only begin its **Part 2** after the parent process has completed **Part 1**.

Synchronization (Rendezvous):

- After completing **Part 1**, each process must wait for the other to finish its **Part 1**. Only then can either process proceed to **Part 2**.