Assignment 1: Multi-Process File Compressor

Objective:
Create a program that compresses multiple files concurrently using fork().

Requirements:

1. The parent process should take a list of files as command-line arguments.
2. For each file, fork a new child process that uses **exec()** to call a compression utility like **gzip.**
3. The parent process should wait for all child processes to finish and report which files were compressed successfully.

Expected Commands:

./compressor file1.txt file2.txt file3.txt

Hints:

- Use fork() to create child processes.
- Use execvp() to run the gzip command.
- Use wait() to ensure the parent waits for all children.

Assignment 2: Process Hierarchy Manager

Objective:
Simulate a process hierarchy like a tree structure using fork().

Requirements:

1. The parent process forks two child processes.
2. Each child process forks two more child processes.
3. The program should display the hierarchy with process IDs in a tree format.

Example Output:

Parent (PID: 1000)

├── Child1 (PID: 1001)

│ ├── Grandchild1 (PID: 1003)

│ └── Grandchild2 (PID: 1004)

└── Child2 (PID: 1002)

├── Grandchild3 (PID: 1005)

└── Grandchild4 (PID: 1006)

Hints:

- Use fork() to create child processes.
- Use getpid() and getppid() to retrieve process IDs.

Assignment 3: Multi-Process Prime Number Finder

Objective:
Write a program to find prime numbers using multiple child processes.

Requirements:

1. The parent process forks N child processes.
2. Each child process finds prime numbers in a different range (e.g., 1-100, 101-200, etc.).
3. The parent process waits for all children to finish and collects the results.

Example Command:

./prime_finder 4

Expected Output:

Child 1: Found primes in range 1-100

Child 2: Found primes in range 101-200

Child 3: Found primes in range 201-300

Child 4: Found primes in range 301-400

Hints:

- Use fork() to create child processes.
- Use wait() to ensure all children completed and cleaned