Task: Process and Thread Management

Overview: In this task, you will learn process and thread management using POSIX and Pthreads. You will implement process-based and threaded solutions, compare their performance, and analyze trade-offs.

Part 1: Process Management

Description: Implement a program with a parent process that creates child processes using fork(). Use message passing with pipes for IPC between parent and child.

Requirements:

- Create child processes using fork()
- Implement IPC using pipes or shared memory
- Manage process execution and termination
- Error handling and cleanup
- Comments and documentation

Part 2: Multithreaded Processing

Description: Implement multithreaded processing using Pthreads. Coordinate threads and access to shared data.

Requirements:

- Create threads using Pthreads
- Coordinate threads to execute a task
- Manage shared access to data between threads
- Manage thread creation, joining/detachment, cleanup
- Comments and documentation

Part 3: Performance Measurement

Description: Compare process and threaded solutions by implementing a parallelizable computational task.

Requirements:

- Implement task (e.g. matrix multiplication) using processes and threads
- Measure and compare execution times
- Vary number of processes/threads
- Present clear performance comparison data

Part 4: Thread Management

Description: Implement joining and detaching of threads. Measure impact on throughput.

Requirements:

- Create joinable and detached threads
- Measure throughput with different join/detach configurations
- Analyze impact of join vs detach on throughput
- Document implementation and analysis

Submission (in C):

- Source code (.c)
- Report (PDF) with explanations, results, analysis and learnings
- Zip and submit by *deadline*

Learning Objectives: Hands-on practice with processes, threads, synchronization, performance comparison.

NOTE: the code for all 4 parts should be implemented in a single C file for submission. Your code is expected to follow the overall structure and flow as below:

- Main function:
  - Fork child process (PART 1)
  - Create threads (PART 2)
  - Implement process vs threaded solutions (PART 3)
  - Join/detach threads (PART 4)
- childProcess function:
  - Contains code for child process (PART 1)
- threadFunc function:
  - Contains code executed by each thread (PART 2)

So in the submitted C file, you should:

- Keep the overall structure provided in main()
- Add actual implementation for child process in childProcess() (PART 1)
- Add thread code in threadFunc() (PART 2)
- Add process vs thread implementations after comments in main() (PART 3)
- Add code for joining/detaching threads after PART 3 in main() (PART 4)