New York University Abu Dhabi
CS-UH 3010: Operating Systems
Programming Assignment 1

Assignment Implementation Techniques

The program involves a sophisticated approach to align with the assignment requirements to achieve parallel computing by distributing the task of finding prime numbers within a given range across multiple processes. It is structured around the concepts of delegators and workers and employs various UNIX system calls, IPC mechanisms, and concurrency management techniques. This document explains the specifics of its design, focusing on key functions and implementation techniques used to accomplish the assignment statement.

The program is comprised of several components: main.c, delegator.c, worker.c, prime.c, and their corresponding header files, delegator.h, and worker.h. It leverages multiprocessing, non-blocking I/O, dynamic signal handling, and effective workload distribution to optimize performance.

Main Component (main.c):

1. **Command-line Parsing:** Uses argc and argv[] to read user inputs defining the lower and upper bounds (lb, ub), the number of worker nodes (n), and the distribution method (even -e or random -r).
2. **Signal Handling:** Implements sigusr1_handler and sigusr2_handler to increment counters upon receiving SIGUSR1 and SIGUSR2 signals to count the number of signals received from the worker.
3. **Non-blocking Pipes:** Employs fcntl to set pipes to non-blocking mode, facilitating efficient IPC without waiting for pipe delegator nodes to complete writing the primes.
4. **Select Function:** Utilizes the select() system call to monitor multiple file descriptors, waiting for one of them to become ready for I/O operations, demonstrating an efficient approach to handling non-blocking I/O.

Delegator Component (delegator.c):

1. **Workload Distribution:** Calculates the start and end points of the range each worker should process. For even distribution, it divides the total range evenly among workers. For random distribution (distribution == 'r'), it randomly assigns varying ranges to each worker while ensuring the entire range is covered.
2. **Forking and Pipe Management:** Uses fork() to create worker processes and pipe() to establish communication channels.

3. **IPC with Workers:** After forking, it redirects the standard output of child processes to the write end of their respective pipes using dup2(), ensuring that the parent (delegator) can capture output from all workers.

Worker Component (worker.c):

1. **Signal Sending**: Communicates with the main process by sending SIGUSR1 or SIGUSR2 using kill(), indicating the completion of tasks or other predefined conditions.
2. **Execvp to Call Prime Computation:** Utilizes execvp() to replace the worker process's memory space with that of the prime.c program. This allows for the modular execution of the prime number calculation.
3. **Time Measurement:** Employs times() system calls to measure CPU and real-time before and after the computation, storing results in a structured array (workerTimes[]), showcasing effective performance monitoring.

Primality Test (prime.c):

1. **Basic Compute Task:** We have utilized algorithm 1, given in the appendix, to obtain the primes given the range after the exec() from the worker node is used.

Performance Optimization Techniques:

1. **Array for Worker Timing:** workerTimes[] in worker.c is a global array storing the execution timing for each worker node. This array allows the main process to compute and display statistics about the computation time, illustrating an effective use of shared data structures for performance analysis.
2. **Non-Blocking I/O and select():** The use of non-blocking I/O combined with the select() system call in main.c enables the program to efficiently manage multiple child processes without idle waiting, optimizing resource utilization.
3. **Dynamic Range Calculation:** The program's ability to dynamically divide the workload among worker processes, especially under the random distribution strategy, demonstrates an innovative approach to load balancing in parallel computing environments.
4. **Dynamic array for sorting:** The program uses a dynamic array for sorting the numbers from the buffer after it has received all the values from the delegator nodes.

This program's design and implementation exemplify advanced programming techniques in UNIX systems, including multiprocessing, IPC, dynamic signal handling, non-blocking I/O, and performance optimization strategies. By judiciously employing these techniques, the program efficiently parallelizes a computationally intensive task.