Homework 2: Dispatcher/Worker Model Solution Description Student IDs: 207477753, 208076919

1. Introduction This project implements a multi-threaded Dispatcher/Worker system in C for Linux. It processes a command file where specific "dispatcher" commands are executed serially, while "worker" commands are offloaded to a thread pool via a shared job queue. The solution prioritizes data integrity and concurrency control using POSIX threads (`pthreads`) and synchronization primitives.

2. High-Level Design The system comprises two main components:

- **Dispatcher (Main Thread):** Parses the input command file, initializes the system, and manages the lifecycle of the worker threads. It handles `dispatcher_msleep` and `dispatcher_wait` commands directly.
- **Worker Pool:** A fixed number of threads (created via `pthread_create`) that consume jobs from a shared queue. Each worker processes complex command lines (containing `msleep`, `increment`, `decrement`, `repeat`) independently.

3. Data Structures

- **Job Queue (`job_queue_t`):** A singly linked list is used to manage pending jobs. Each node (`job_t`) contains the command string and the timestamp of when it was read (`read_time_ms`), allowing for accurate turnaround time calculations.
- **Counter Files:** Integers are stored in persistent text files (`countXX.txt`).
- **Logging:** Trace data is written to `dispatcher.txt` and `threadXX.txt` to track execution flow.

4. Synchronization & Concurrency To ensure thread safety and maximize parallelism, the following mechanisms are used:

- **Queue Protection:** A global mutex (`queue_mutex`) protects all operations on the `work_queue` (enqueue/dequeue).
  +1
- **Condition Variables:**
  - `queue_not_empty`: Signals sleeping workers when a new job is available, preventing busy waiting.
    +2
  - `all_jobs_finished`: Signals the dispatcher when the queue is empty and all active workers are idle, enabling the `dispatcher_wait` command.
    +1
- **Fine-Grained File Locking:** An array of mutexes (`file_mutexes[MAX_COUNTERS]`) is used to protect counter files. This allows multiple workers to update *different* counters

simultaneously without blocking each other, while ensuring atomic updates for the *same* counter.
+1

- **Statistics Protection:** A `stats_mutex` ensures that updates to global statistics (sum, min, max turnaround times) are atomic.
  +1

5. Implementation Details

- **Command Parsing:**
  - The dispatcher uses `strtok` to parse the main command file.
  - Worker threads use the reentrant function `strtok_r` to parse job lines safely without interfering with other threads' string processing.
- **File Operations:** Counters are updated using a read-modify-write cycle: `fopen` -> `fscanf` -> `rewind` -> `fprintf` -> `ftruncate` -> `fclose`. This ensures the file content remains clean.
  +1
- **Memory Management:** The system dynamically allocates memory for job nodes and the worker thread pool. A comprehensive cleanup routine runs at the end of the `dispatcher` function to `join` all threads, `free` allocated arrays, and `destroy` all mutexes and condition variables