Homework 2: Dispatcher/ Worker model with Linux pthreads

Submitted by: Sagi Ber (208276188), Rotem Refaeli (322281775), Roy Porat (205947690)

Overview

This program implements a dispatcher/worker model using POSIX threads. The dispatcher reads commands from an input file and either executes them sequentially or offloads them as jobs to worker threads, which process them concurrently. The dispatcher uses a shared queue to manage jobs efficiently, ensuring proper synchronization between threads. The program prevents race conditions by mutexes and condition variables, enabling thread-safe operations and minimizing CPU usage.

Api

- **#include <stdlib.h>:** Provides general-purpose standard library functions for memory management, random numbers, conversions, etc.
- **#include <pthread.h>:** Provides functions for managing POSIX threads (multi-threading).
- **#include <stdio.h>:** provides functions for input/output operations.
- **#include <string.h>:** Provides functions for manipulating strings and memory blocks.
- **#include <time.h>:** Provides functions to handle and manipulate time.
- **#include <unistd.h>:** Provides access to the POSIX operating system API.
- **#include <limits.h>:** Defines constants for variable limits (LLONG_MAX, LLONG_MIN).

#define Constants

Name	Value	Purpose
MAX_COUNTER_SIZE	100	Represents the maximum number of counter files (countxx.txt)
MAX_THREAD_SIZE	4096	Specifies the maximum number of worker threads that can be created.
MAX_LINE_SIZE	1024	Defines the maximum allowed length of a command line in the input file.
CLOCK_REALTIME	0	Indicates the clock ID used with clock_gettime to retrieve the current system time.
MAX_QUEUE_SIZE	4096	Specifies the maximum number of jobs that can be queued in the shared queue.
MAX_JOBS	4096	Represents the maximum number of jobs that can be processed by the system.

Files Generated

1. **Counter Files (countxx.txt):** Stores counter values updated by worker threads.
2. **Worker Logs (threadXX.txt):** Logs actions performed by worker threads.
3. **Dispatcher Log (dispatcher.txt):** Logs actions performed by the dispatcher.
4. **Statistics (stats.txt):** Summarizes runtime and job turnaround metrics.

Code Structure

1. Initialization

- **initialize_queue:** Initializes the job queue, mutex, and condition variables.
- **create_counter_files:** Creates and initializes counter files (countxx.txt) to 0.
- **create_threads:** Creates num_threads worker threads.

2. Dispatcher Logic

- **dispatcher:**
  - Reads commands from the input file.
  - Executes dispatcher-specific commands (dispatcher_wait, dispatcher_msleep).
  - Offloads worker jobs to the work queue.
  - Signals workers when the input file is fully processed and waits for their termination.
  - **dis_wait:** Makes the dispatcher wait until all active jobs are completed.
  - **dis_msleep:** Sleeps for a specified time in milliseconds.
  - **worker:** Offloads a job to the worker threads by **enqueuing** it to the shared work queue.

3. Worker Logic

- **worker_function:**
  - Waits for jobs in the job queue or termination signal.
  - Executes offloaded jobs, logging their status and processing statistics.
  - Signals the dispatcher when all jobs are done.
  - **dequeue:** Removes jobs from the shared work queue.
  -

4. Job Processing

- **process_job:** Handles different job types:
  - **msleep x:** Sleep for x milliseconds.
  - **increment x:** Increment the value in the counter file countxx.txt.
  - **decrement x:** Decrement the value in the counter file countxx.txt.
  - **repeat x:** Repeats a sequence of commands x times.

5. Log files

- **log_worker:** Logs worker activity to threadXX.txt files.
- **dispatcher:** Logs dispatcher actions to dispatcher.txt.

6. Statistics

- **print_statistics:** Writes them to stats.txt.

Compilation and Debugging

All input files and command-line arguments are assumed to be valid. The code was compiled using the GCC compiler with strict warning flags enabled. Additionally, the program was thoroughly debugged using GDB to trace execution flow, identify segmentation faults, and analyze thread behaviour, ensuring correctness and reliability.

Summary

This project provides hands-on experience with threads and the dispatcher mechanism. It teaches us how to design a dispatcher and work with the pthread library, emphasizing synchronization and thread safety. We learn how to avoid race conditions and other common threading issues, building a deeper understanding of multithreaded programming concepts.