Documentation for HW 2 - Dispatcher/Worker Model With Linux Pthread

Submitted by: Nicole Frumkin (ID 211615372) and Yaron Silberstein (ID 318321759)

Overview

This program includes a dispatcher implementation that processes commands that are read from an input file (cmdfile.txt) and places them in a job queue. It supports multiple worker threads that access this queue and are synchronized using mutexes and condition variables. The number of threads is defined by the user and the maximum threads allowed are 4096. The program executes worker commands like 'increment' (counter file), 'decrement' (counter file), and 'msleep', tracks statistics, and generates log files for analysis. The maximum number of counters is 100 and also defined by the user as an input. The program also supports dispatcher wait, where the program waits until all background commands have finished before processing next lines.

Code Structure and Functions

Header File

The header file includes all the standard libraries required for the code to run properly and help us implement the functionality, Structs and all functions declarations. The libraries are: input/output operations (<stdio.h>), memory management (<stdlib.h>), string manipulation (<string.h>), logical operations (<stdbool.h>), time handling (<sys/time.h>), and operating system-level functions (<unistd.h>).

It also uses <pthread.h> for multithreading, enabling thread creation, synchronization with mutexes, and condition variables.

The file defines constants like MAX_THREADS and MAX_JOBS to set program limits and structures such as JobQueue, a thread-safe queue for managing tasks, and WorkerArgs for passing thread-specific data - such as TID.

Our functions handle queue management, thread creation, command execution, and logging, and global variables and mutexes ensure thread-safe operations and enable tracking of performance metrics, such as turnaround times and job counts.

Main Function

The main function implements the dispatcher. First, it checks if the number of arguments is correct (namely, equals 5). Then, it initializes resources (such as job queue, number of counter files to create and more). It creates the wanted amount of worker threads and begins to read job commands line by line. In the end, it announces a shut down signal to the treads to make them exit the infinite loop and the dispatcher waits for all jobs to complete and cleans up resources.

worker_thread Function

This function runs on each worker thread and processes jobs dequeued from the job queue. It handles special commands like 'repeat' to execute commands multiple times. The function also updates statistics for turnaround times and ensures safe access to shared resources using mutexes.

create_counter_files Function

This function creates the counter files in case they do not exist. The function reset the value to 0 for each counter, and closes the file.

create_threades Function

This function creates num_threads threads, and allocates memory for them. It also sets the unique worker I.D. and signals in case of a memory or thread creation error.

read_lines Function

This function iterates over the input commands file, reading it line by line. In case we have a dispatcher command, it executes it on the fly, and in case it is a worker function, the line is being added to the queue and the worker_thread function will execute it.

init_queue Function

Initializes the job queue with front (index of the first job), rear (index of next free slot), count (number of jobs in the queue), and shutdown status (if we stopped reading lines). It also sets up mutexes and condition variables for synchronization between the dispatcher and worker threads. We defined it in the queue structure for easier access and to avoid global variables.

enqueue and dequeue Functions

- **enqueue:** Adds a job to the circular job queue in a thread-safe way, making sure other threads do not access the queue if the dispatcher adds a job to it. If the queue is full, the dispatcher waits using a condition variable. When workers dequeue a job and the queue has space, we broadcast the dispatcher to wake him up.
- **dequeue:** Removes a job from the queue in a thread-safe way. If the queue is empty, the thread sleeps until new jobs are available or shutdown is broadcasted.

execute_command Function

Each worker parses the job command to execute it:

- increment <counter>: Increments the value in a counter file. Making sure to increment the value within the file each time and not use a helper array to update the value only at the end of the program.
- decrement <counter>: Decrements the value in a counter file.
- msleep <time>: Thread sleeps for the specified duration in milliseconds.

It uses mutexes to ensure thread-safe access to counter files.

create_thread_files Function

The function creates the right amount of thread log files if the log_enabled is set to 1.

get_current_time_in_miliseconds Function

The function returns a long long type of the current time in milliseconds. It is used to calculate time differences during the program run.

print_to_log_file Function

Logs job execution details to a log file for each worker thread.

create_stats_file Function

Generates a 'stats.txt' file summarizing:

- Total runtime.
- Sum, minimum, average, and maximum job turnaround times.

trim Function

Cleans up and handles the spaces in strings. It is being used to write commands to log files.

Output Files

- countXX.txt: Counter files storing values for increment and decrement operations.
- threadXXXX.txt: Log files for each worker thread (if logging is enabled).
- dispatcher.txt: Log file for dispatcher activity.
- stats.txt: Summary of runtime and job turnaround statistics.