Operating Systems

Homework 2: Dispatcher/Worker model with Linux pthreads

Documentation

Submitters:

Hagai Mozes - 205562010

Shanie winitz - 316295112

Main flow: (the dispatcher)

1. In this part we create dispatcher log file if needed, according to the argument "log_enabled" given by the user.
2. The dispatcher creates all the counter files, according to the argument "num_counters" given by the user. It also creates lockers for the files, that we use later in order to avoid that few threads will try to access a file in parallel.
3. The dispatcher also creates all the threads, according to the argument "num_threads" given by the user. The creation of the threads is done using the worker_thread function that we elaborate about it later.
4. Then, the dispatcher reads the command file line by line. For each line, we check if it is a dispatcher_msleep command, a dispatcher_wait command, or a worker command, and operate accordingly.
    a. dispatcher_msleep: using usleep() to wait for the wanted time.
    b. dispatcher_wait: We lock threads_mutex and then we check whether there are running threads or jobs in the queue, and use pthread_cond_wait to wait for all the jobs to finish. Then We ulock threads_mutex.
    c. worker command: is handled with the function parse_worker_line that we elaborate about later.
5. When reading the line is finished, we raise a flag so that the worker_thread function will operate correctly.

6. Before finishing the program, the dispatcher destroys all the thread locker's conditions, the thread lockers, the dispatcher locker's conditiond, the counters' lockers, the queue locker. It also closes all the files, and creates the statistics file.

worker thread function:

This is the function for the creation and operation of the threads.

1. we want the threads to be alive for the entire run of the program, and if a thread has no work to do at some point, we want it to go to sleep until new work is available to pick up. So , at the begging, when a thread is created, we lock the thread. Then we used a while loop checking whether there are jobs in the queue, and if the reading of the cmdfile is not finished – if so, we wait on the condition using pthread_cond_wait(&cond_threads, &threads_mutex); and the queue is unlocked. In this part we also keep count of how many running threads are there and send a signal to dispatcher_wait to indicate when all commands finished.
2. When exiting the while loop, we unlock the thread.
3. If there are no more jobs in the queue, and we finished reading the cmdfile, we unlock the queue and break the while loop.
4. Otherwise, when there is a job in the queue to be picked up by the thread, we dequeue it and execute the commands in the line. Then we free space in the queue.
5. If log_enable is on, then a file for the thread's loggings is created.

Other Functions:

1. write_log_line – write to a log file the current job that is being executed, according to mode (start or end).
2. update_calc - update worker statistics.
3. parse_worker_line – used when the current job is a worker job, to separate it to different arguments for execution (whether it is a msleep/ increment/ decrement command, which counter, do we need to repeat…). We save the product in a linked list, that will be used by the thread function for execution in the threads.

4. action_on_counter – the function gets the counter arguments (a node in the linked list) and uses them to increment or decrement the value in the counter file. Before accessing to the file we use pthread_mutex_lock(&counters_mutex[curr_counter_args->cmd_num]) to prevent different thread to try access the file. We read the file content, increment or decrement the value, Afterwards, we close the file and use pthread_mutex_unlock(&counters_mutex[curr_counter_args->cmd_num]); so that other thread could access the file now.
5. create_stats_file – creating a file for the ststistics of the program, and print to it the total running time, sum of jobs turnaround time, min job turnaround time, average job turnaround time, max job turnaround time.

Queue.h

We created another c file that will hold all the queue handling functions, like dequeue, enqueue, free linked list, print linked list.

Here there is also a function using to concatenate the linked list "repeat" number of times, when needed.

global variables:

we used global variables when there was a need for different threads accessing the same variables. For example, global lockers, global locker's conditions, "num_running_threads", for keeping track of how many busy threads are there in a certain moment, finish flag for indicating that the command file read is finished, variables for the statistics, etc...