Ming Tai Ha: mingtha
Harsh Kothari: hk596
Zheng Yu: zy120

Assignment 1: Multilevel Priority Queue

In this assignment we implemented a Multilevel Priority Queue, which we will abbreviate as MLPQ. The MLPQ follows the requirements given in the assignment.

Structure

`The structure of the MLPQ is decomposed as follows:

1) The queue struct is the common struct used for the running queues and wait queues.

2) The scheduler struct is composed of running queues, where there are as many queues as there are priorities. The level of the highest priority is 0, and the lowest is 15. (In our implementation we have set the number of running queues to be 16, but this value can be easily changed.) We have also specified the number of locks which our pthreads library allows to be 5. The time quanta of the each level are linearly defined, and the time slice of each queue is (QUEUE_PRIORITY_LEVEL+1) * 50ms.

3) The pthreads and mutex requirements are written as required by the homework description. Namely, threads are added to the scheduler when the pthreads are created. Each mutex has a waiting queue. When my_pthread_mutex_lock is called and the mutex has been preempted by another thread, my_pthread_mutex_lock will put the calling thread to its waiting queue and call the scheduler. When the lock holder finished its work in the critical section, it will call my_pthread_mutex_unlock and pick one thread from the waiting list, then put it back to running list.

4) The scheduling takes place in the schedule_handler() function. This function is called after the 50ms quanta has expired. Before the scheduler checks the queues, the scheduler clears the timer. Then the scheduler first checks if there are any thread which have pass the AGE_THRESHOLD (maintainence cycle of 2 sec), going through every queue except the one with highest priority; this takes place every scheduling cycles. Afterwards, the scheduler will check whether the current running thread has run its full time slice. If so, we pick out another thread from a scheduler and demote the priority of the previous thread; if not, we continue running the thread. The length of the time slice is defined by the priority of the queue. Then we reset the timers swap the context. Also, whenever a thread yields, finishes, or calls the exit function, it will call the schedule_handler. When the mutex is blocked, the blocked thread will also called the scheduler.

Testing and Results

We have tested the scheduler using the following function: Run a nested for-loop, which runs from 10^4 to 10^10 total iterations. The values for the iterations range from 10^2 to 10^5, and were randomly chosen so that we can get a mixture of threads with varying load sizes. This was chosen so that the threads running the function will run for more than 50ms and allow us to see how the scheduler evolves. We compared our MLPQ to a Round Robin scheduler by measuring the response time and the total and average run times for 10, 20, 30, 40, and 50 threads. The following table details our result. We found that the response time of each thread is about 50ms * (n – 1), where n is the position of the thread on the queue relative to the front of the queue for both the Round Robin and MLPQ. This is expected for Round Robin by definition and for our MLPQ, which uses MLPQ for the priority of each queue. We have, however, found that the total and average runtime of all the threads to be smaller using MLPQ than using Round Robin. The following table details our results, and we see that the average runtime for MLPQ is smaler than the runtime of Round Robin by at least a factor of 2. Thus, we find that the MLPQ has similar response time for each thread submitted.

Table I. Run Time information with respect to the Number of Threads

Round Robin					
Threads	10	20	30	40	50
Runtime (s)	85.273702	143.964785	207.536631	301.885825	275.227194
Avg Runtime (s)	8.5273702	7.19823925	6.9178877	7.547145625	5.50454388
MLPQ1					
Threads	10	20	30	40	50
Runtime (s)	30.218771	52.796329	52.796329	43.708061	158.144928
Avg Runtime (s)	3.0218771	2.63981645	1.759877633	1.092701525	3.16289856
MLPQ2					
Threads	10	20	30	40	50
Runtime (s)	21.915794	16.638953	45.195366	27.304339	90.928302
Avg Runtime (s)	2.1915794	0.83194765	1.5065122	0.682608475	1.81856604
MLPQ2					
Threads	10	20	30	40	50
Runtime (s)	51.37294	13.514256	43.945468	52.431063	169.406811
Avg Runtime (s)	5.137294	0.6757128	1.464848933	1.310776575	3.38813622
MLPQ Avg					
Threads	10	20	30	40	50
Runtime (s)	34.50250167	27.649846	47.31238767	41.147821	139.493347
Avg Runtime (s)	3.450250167	1.3824923	1.577079589	1.028695525	2.78986694

We also tested how the Round Robin and MLPQ handles locks. We used a test case by running a function which modifies a shared variable, where one thread adds while another thread subtracts from the shared variable. We performed this for 10 threads to see how the different schedulers perform. We find that for 10 threads, the run time for the MLPQ scheduler is less than that of Round Robin.

Table II. Run Time information using locks with 10 threads

Locks	MLPQ	Round Robin
Runtime (s)	11.762887	31.606095