Project 2 - A task scheduler simulation

CSE 503 – Summer 2024

Richard Douglas

Assignment Description

We were asked to edit code in a file called hash.cpp containing an incomplete implementation of singly linked list (with a typo). Specifically, we were asked to do:

File input.txt contains 100,000 numbers separated by new line. Do the following:

1.1 Write a C++ program to compute the summation of these 100,000 numbers using single thread.

1.2 Write a C++ program to compute the summation of these 100,000 numbers using 10 threads, meaning each thread compute 10,000 summations and the main thread sum the result of the 10 threads.

1.3 Use the time syscall to compare the time spent for 1.1 and 1.2

Logic Employed

I used a singly linked list to store all of the lines from the input text file as string objects. This stored the lines in reversed order, which was perfectly fine to my mind. I just parsed the list in that (reversed) order, tokenizing the string into an integer array of three values which I would subsequently assign to a new object of a custom “Job” class. At this point in the single pass through the forward linked list holding the string values from the input file, I enqueue the new Job object into a custom JobQueue class object. The JobQueue is… well… a queue data structure. In other words, another singly linked list. However, I did not use the STL implementation and instead created my own class.

Now that I had populated my queue with Job objects, I used a while loop to “run” all of the jobs in succession, until completion. The condition of the while loop is whether or not my “isEmpty” member function of the JobQueue class evaluates to true. To ensure Jobs were not run before their “time_in” or “requesting time”, the first piece of logic in my while loop was a conditional statement assessing whether or not this job’s requesting time was greater than a global variable called “the_time”, which represented the time elapsed since the program started…. In hindsight, I suppose I could have used the difference in a timestamp in milliseconds to more realistically simulate time elapsing while jobs “run”... but I didn’t… and I’m not gunna because it was not specified to do so in the project instructions and I’m feeling a bit over it. Anyways, the_time is arbitrarily set to the integer 0 and is increased by an integer representing the duration of each job’s CPU burst. Thus, all of the jobs can be run! The first job in the queue is accessed and its duration is decremented by the time quantum. the data members of this job are then assigned to a new job object. The job is then dequeued and if its duration is still above the time quantum, it is enqueued back to the JobQueue.

This iterative process is repeated and appropriate output is printed to the terminal until all of the job durations are equal to or less than 0. Thus, all of the jobs are completed.

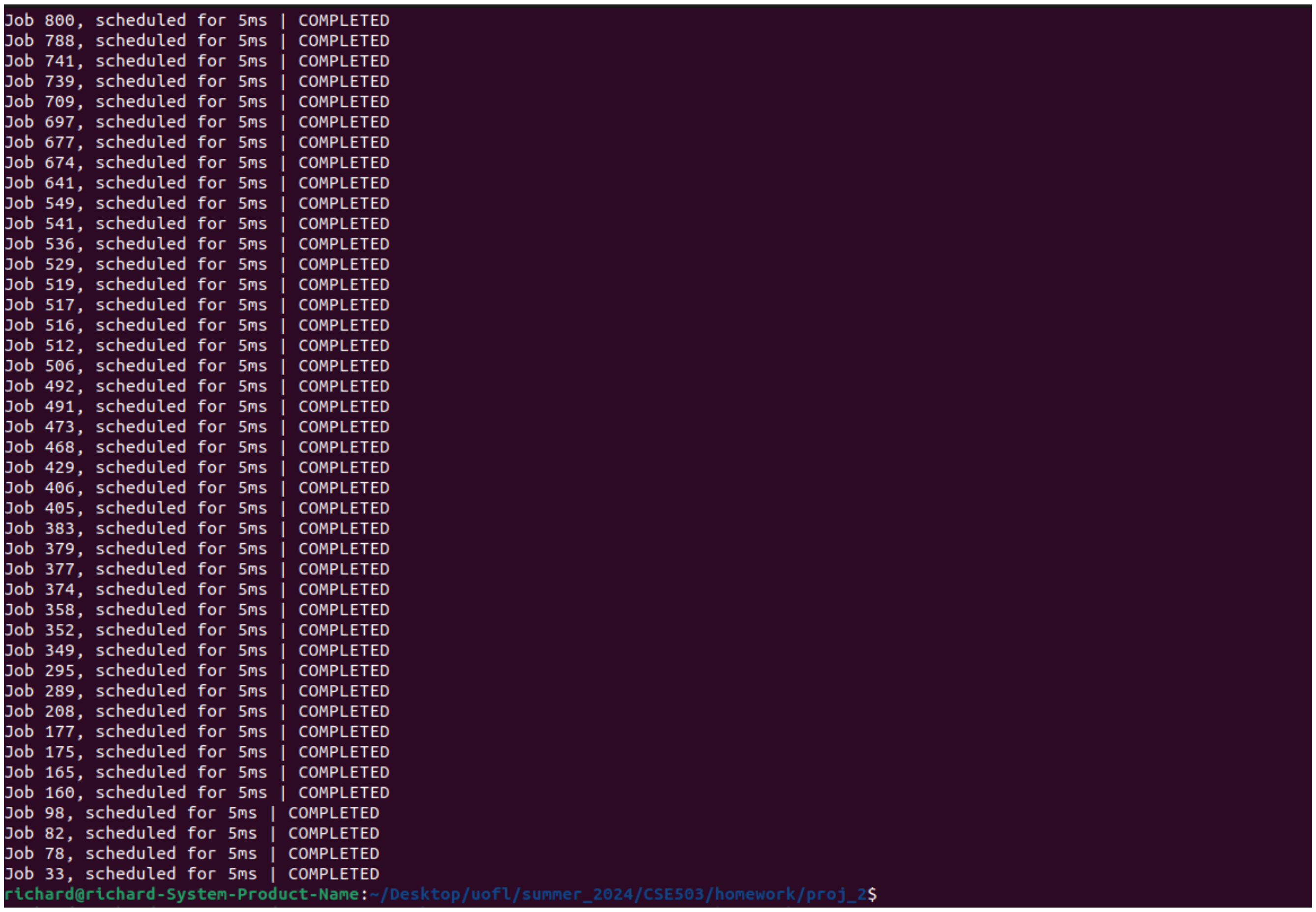
I/O

Input:
Job.txt is provided as an input file. Each line contains the name of the job, requesting time (millisecond since the computer started) and duration (millisecond).
For example: Job 1, 20, 50 means a job for CPU called “Job 1” is requested to run at 20ms since the CPU started, and the duration of the job is 50ms.

Output to console showing the job number, the amount of time for which it was scheduled to run. I believe all job durations were multiples of 5 or 0, so it was always 5 unless 0… which kind of messed with me at first. I thought to myself…. Surely, some of the jobs aren’t multiples of five. Is something wrong with my code? but no…. They were all multiples of five.

Screenshots

The following is the output run in my terminal.



```
Job 800, scheduled for 5ms | COMPLETED
Job 788, scheduled for 5ms | COMPLETED
Job 741, scheduled for 5ms | COMPLETED
Job 739, scheduled for 5ms | COMPLETED
Job 709, scheduled for 5ms | COMPLETED
Job 697, scheduled for 5ms | COMPLETED
Job 677, scheduled for 5ms | COMPLETED
Job 674, scheduled for 5ms | COMPLETED
Job 641, scheduled for 5ms | COMPLETED
Job 549, scheduled for 5ms | COMPLETED
Job 541, scheduled for 5ms | COMPLETED
Job 536, scheduled for 5ms | COMPLETED
Job 529, scheduled for 5ms | COMPLETED
Job 519, scheduled for 5ms | COMPLETED
Job 517, scheduled for 5ms | COMPLETED
Job 516, scheduled for 5ms | COMPLETED
Job 512, scheduled for 5ms | COMPLETED
Job 506, scheduled for 5ms | COMPLETED
Job 492, scheduled for 5ms | COMPLETED
Job 491, scheduled for 5ms | COMPLETED
Job 473, scheduled for 5ms | COMPLETED
Job 468, scheduled for 5ms | COMPLETED
Job 429, scheduled for 5ms | COMPLETED
Job 406, scheduled for 5ms | COMPLETED
Job 405, scheduled for 5ms | COMPLETED
Job 383, scheduled for 5ms | COMPLETED
Job 379, scheduled for 5ms | COMPLETED
Job 377, scheduled for 5ms | COMPLETED
Job 374, scheduled for 5ms | COMPLETED
Job 358, scheduled for 5ms | COMPLETED
Job 352, scheduled for 5ms | COMPLETED
Job 349, scheduled for 5ms | COMPLETED
Job 295, scheduled for 5ms | COMPLETED
Job 289, scheduled for 5ms | COMPLETED
Job 208, scheduled for 5ms | COMPLETED
Job 177, scheduled for 5ms | COMPLETED
Job 175, scheduled for 5ms | COMPLETED
Job 165, scheduled for 5ms | COMPLETED
Job 160, scheduled for 5ms | COMPLETED
Job 98, scheduled for 5ms | COMPLETED
Job 82, scheduled for 5ms | COMPLETED
Job 78, scheduled for 5ms | COMPLETED
Job 33, scheduled for 5ms | COMPLETED
richard@richard-System-Product-Name:~/Desktop/uofl/summer_2024/CSE503/homework/proj_2$
```

This is always the output, since round robin is a deterministic algorithm and the input is known and unchanging.

Conclusions

This was a little bit more difficult of a concept. I’ve always found threading a little bit mysterious and fun, so it was good to get in some practice. It really makes you think about the logical architecture of a modern computer. Pretty fascinating stuff.