Chandler Teigen
CPTS 223
PA5 report

Diagram of Scheduler system:

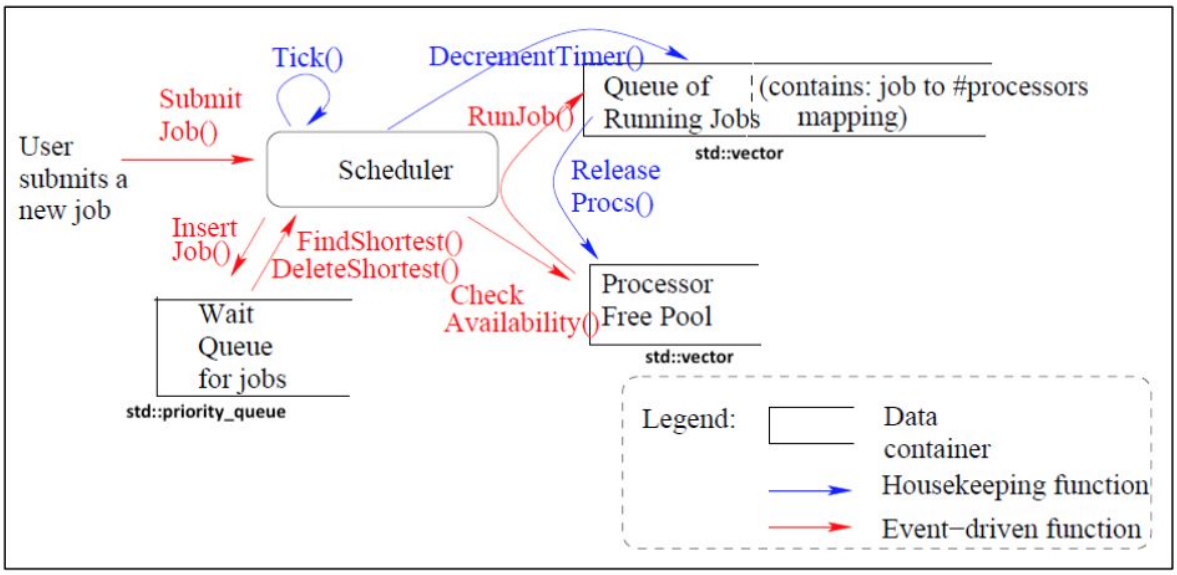

Figure: borrowed from the assignment, with updated information on the STL containers used to implement each data container in the scheduler.

The algorithm that I used was made to follow the assignment guidelines as closely as possible. I used an std::priority_queue to implement a min heap in order to store the jobs in the waiting queue. I made this choice because the smallest element was always needed next in this scheduler, which a min heap can do in O(1) time.

I implemented the running jobs queue as a simple std::vector. The reason for this is because of the constant need to update the values of the ticks_left member of each Job in the queue. The vector allowed for me to iterate over the elements and update each one in O(n) time. It seemed like a good idea to use a min heap for this as well, but I found that updating every element would require more time complexity because of the need to change each value and use STL functions to retain the heap requirements.

Finally, I implemented the processor pool as a vector as well, because the number of processors is most likely going to be very low compared to the number of jobs, so taking the time to sort the processors isn't worth it because no matter what, the container will have to be iterated over to find the correct processors to assign and release.

Function	Worst case run-time complexity
insert_job	O(logn)
find_shortest	O(1)
delete_shortest	O(logn)
check_availability	O(1)
run_job	O(p) (assigning processors)
decrement_timer	O(n) (n = running jobs only)
release_procs	O(p)

Bottlenecks/shortcomings

In my opinion, the biggest shortcomings of the shortest-first strategy are that the long jobs may never be completed and that the processors max number of processors is not always used. First, if a long job is put into the waiting queue, but smaller jobs keep being added to both the waiting queue and the running queue, the long job may have to wait a very long time to be processed even though it was queued long before the shorter jobs. The longer job could be more important than the smaller jobs as well.

The other shortcoming that I noticed was that the processors could be not all used at once. The algorithm only every checks if there are enough processors available for the shortest job, so if the shortest job requires a large number of processors, then the number of processors being utilized may not be very high. For example, if the only job being run had n_proc=1 and n_ticks=10, and the shortest job required the max number of processors, then the scheduler would be forced to wait 10 ticks before it could assign the processors to the shortest job. Meanwhile, other jobs that require less processors could have been assigned while the other job was running.