Albert Wu
CptS 223
Design Document

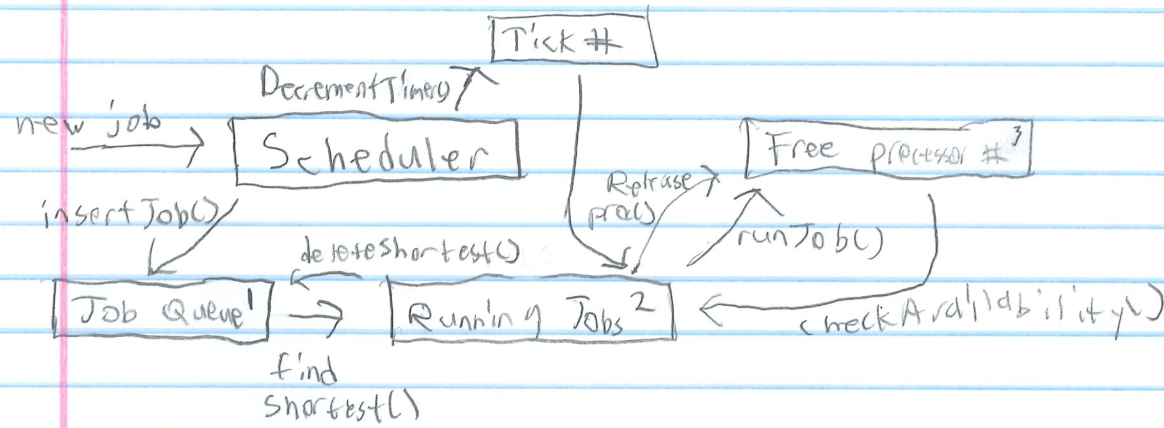
See second page for diagram.

*note: I tracked the current "time" (tick number) as a field of my Scheduler class. Thus, I was able to use a hashmap with the completion time of each job as keys and the jobs themselves as values to store the running jobs. Thus, at each tick I was able to remove each completed job in constant time per job.

Worst-case complexities:
n is the total number of jobs.
m is the maximum number of jobs completing at the same tick.

insertJob()	O(log n)
findShortest()	O(1)
deleteShortest()	O(log n)
checkAvailability()	O(1)
runJob()	O(1)
decrementTimer()	O(1)
releaseProcs()	O(m)

The main shortcoming of this shortest-job-first strategy is that it completely ignores how many processors the job takes. For example, if there are only 5 processors available, and job A takes 5 processors and 10 ticks, and job B takes 6 processors and 9 ticks, our strategy will wait until at least 1 more processor becomes available in order to be able to run job B. Job B becomes the bottleneck, causing our strategy to waste 5 processors for an indefinite amount of ticks. If no processors would become available for 10 ticks, an optimal strategy would have completed job A before our strategy even starts job B or job A. Clearly our strategy is suboptimal in terms of both performance and functionality.



1 std::make_heap()
2 std::unordered_map
3 integer number