

Kace Cottam (11609561)
 Maia Whitley (11625970)
 Programming Assignment 5
 CPTS223.01

Programming Assignment 5

Scheduler System Design

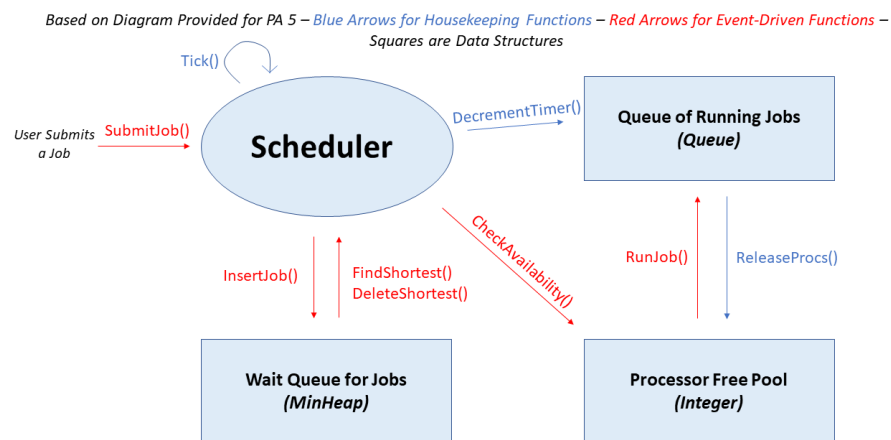


Figure 1:

Runtime Complexities

Function	Complexity
InsertJob	$O(\log n)$
CreateJob	$O(1)$
ReadJob	$O(n)$
FindShortest	$O(1)$
DeleteShortest	$O(\log n)$
CheckAvailability	$O(1)$
RunJob	$O(1)$
DecrementTimer	$O(n)$
ReleaseProcs	$O(1)$

InsertJob - Insert into job_queue

CreateJob - Just makes checks with parameters

ReadJob - Depends on how many characters we have to read or ignore.

FindShortest - Because we are using a minheap, the shortest job is the first value.

DeleteShortest - Because we are using a minheap, delete the shortest value and percolate

CheckAvailability - It just compares values, so $O(1)$

RunJob - it adds to the end of a vector

DecrementTimer - Needs to visit all values in running_jobs

ReleaseProcs - just a subtraction and removal from vector

Short-Comings of Shortest-Job-First Strategy

One of the major disadvantages of the shortest-job-first strategy is that it's optimizing jobs purely based on which one takes the least amount of time to run. This overlooks the importance of the number of processors, the length of time a job has been in the queue, and the importance of a job. For example, a job that was just inserted that takes 4 ticks and uses 20 processors would always be prioritized over a job that has been waiting in the queue, takes 5 ticks, and uses 1 processor. This could result in the inefficient use of resources and longer jobs could wait in the queue indefinitely if shorter jobs continue to be added.

From a performance standpoint, the shortest-job-first strategy doesn't seem to offer a significant benefit over a first-come-first-serve strategy and adds additional time complexity for jobs to percolate within the priority queue as opposed to jobs simply being added to the end of a regular queue for a first-come-first-serve approach.