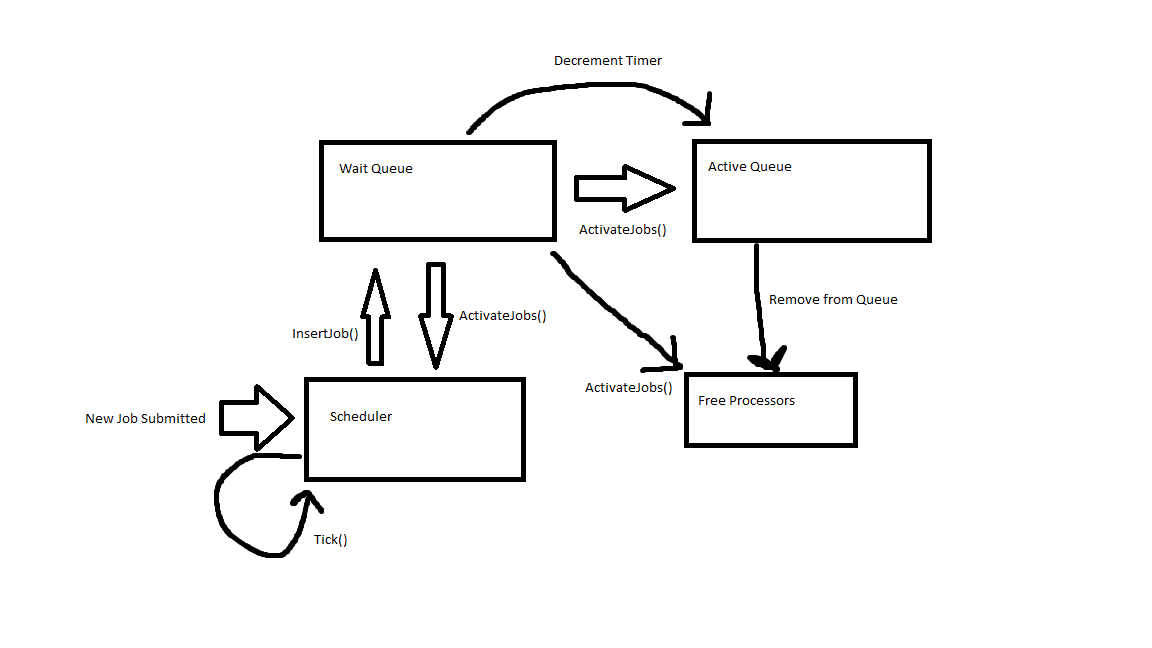
Kyle Newton

Programming Assignment 5

Design Document



Runtimes:

|  |  |
| --- | --- |
| Function/Idea | Worst Case Runtime |
| InsertJob() | O(log(n)) |
| FindShortest() | O(n) |
| DeleteShortest() | O(2log(n)) |
| CheckAvaliablility() | O(1) |
| RunJob() | O(n) |
| DecrementTimer() | O(1) |
| ReleaseProcs() | O(1) |

Paragraph:

I believe that the greatest shortcoming of using the shortest-job-first strategy is that there may be more intensive jobs will get shunted to the back of the queue, only to be seen once all of the shorter jobs were finished. This can be an issue that causes important jobs to be ignored in favor of shorter, less vital ones. In an enterprise situation, this can lead to loss of productivity if there is always something being waited on. The performance of this also takes a hit because you are more likely to have a lot of small jobs running at once, rather than one or two large jobs, which may in the end take up more resources. I feel as if the best way to completely implement this would be to keep two processor pools, one of which is reserved for workloads larger and then all jobs requiring less processors than that are relegated to the other processors. However, in implementation, I’m not very sure how this would work and it would be a lot of trial, error and math before finding the correct way to implement this.