**Algorithm Design:**

I intend to implement a scheduling algorithm that adopts a methodology very similar to “Shortest Job First” so that jobs with the shortest estimated processing time by the SAC-SimOS will be processed first in order to minimize the average waiting time of each individual process that the synthetic CPU will be responsible for. However, I intend to build my algorithm in a preemptive manner so that my CPU scheduler can effectively multitask by switching and processing multiple processes in a very short amount of time.

**Algorithm Pseudocode:**

1. Create a MinHeap Priority Queue Class
   1. Construct the Priority Queue in a way so that it automatically determines what the position of each new process in the queue should be based on whether its estimated completion time is shorter than other processes already in the queue
2. Instantiate a new Priority Queue object in the new CPU\_Scheduler class
3. Create CPU\_Scheduler Class Constructor that implements the pre-made Process Scheduler Interface
4. Create an “addProcess(SimProcessInfo process)” Method that is responsible for adding new processes to the new Priority Queue structure based on how quickly they processed by the SAC-SimOS
   1. Ensure that it will automatically sort through all elements and prioritizes those that have the shortest estimated completion time as their Keys so that they will be run by the SAC-SimOS first (thus minimizing the average waiting time for processes sitting in the queue)
   2. Add the new process to the MinHeap Priority Queue accordingly
5. Create a “getNextProcess()” Method that is responsible for returning the p\_id (as an integer) for the next scheduled process that SAC-SimOS will run
   1. While there are still processes left in the queue and the state of the process at Index 0 of the queue is equal to “Terminated”, remove the process at Index 0 from the queue
   2. Check to make sure that there is another process in the queue that needs to be run
      1. If there isn’t another process, return “-1”
   3. Decrement the “Key” of the Process at Index 0 in the Priority Queue by 1 to indicate that the process at Index 0 requires one fewer clock cycle (and therefore has less estimated computational time remaining) so that new processes being added to the queue can be added appropriately based on their total estimated computation time compared to the remaining computation time of the processes already in the queue
   4. Return the p\_id of the next process at Index 0 in the Priority Queue

**Algorithm Analysis:**

* *What do you take into account when considering priority?*

My algorithm prioritizes jobs with shorter estimated completion times in order to minimize the total average wait time of other processes just sitting idly in the queue. My algorithm also makes a concerted effort to update the estimated computation time of each process in the queue waiting to be processed, allowing it to be even more efficient.

* *How does your algorithm treat high priority processes (i.e. what does it do differently)?*

My algorithm elevates those processes with higher priority (those with shorter estimated completion times) to the beginning of the MinHeap Priority Queue so that they will be processed sooner, thus decreasing the average wait time for all processes.

* *Prove whether or not starvation is possible with your scheduling algorithm.*

Starvation is possible with my algorithm because processes that are estimated to have longer remaining completion times will always get pushed down towards the end of the Priority Queue. However, the tradeoff to this is that more processes with shorter estimated completion times will be run by the SAC-SimOS operating system sooner, which in turn minimizes the overall average wait time for processes being scheduled by my algorithm. In other words, unlike a Round Robin implementation for CPU scheduler, my algorithm fully embraces the idea that it’s not going to treat every process equally in order to pop smaller jobs from the queue faster.