Shaun McThomas
Student ID: 13828643
CS 143A
06/10/2015
Report on Process Schedulers project

I started this project realizing that a container was necessary to hold each process details. This is why I started by creating the Process class. This class holds: the process ID, the arrival time, last time the process used the CPU (based on a system clock I would decide on in the schedulers), the turnaround time, the starting burst time, the shares, the remaining burst time, and the current wait time. This class included a "runFor" method which took in the system time and the length to run the process as parameters. It returns what the system time should be after the process has ran for the specified amount. This is used to simulate the process running on the CPU. I also created some output functions for debugging and final output. Every action in this class runs in constant time, making every action O(1).

After completing the process class, I moved onto the FCFS scheduler. First thing I did was read all processes in from the input file and store them into a priority queue. For this I created a comparator class for the process class to compare processes based on arrival time. To read and store all processes into this arrival queue requires n insertions into the priority queue. One insertion is done in O(log n). Therefore the overall run time complexity for reading in and storing the processes into the arrival priority queue is O(n log n). (This method was repeated for all schedulers and will only be discussed here.) After having these processes stored in this arrival queue, I began taking them out of the queue then calling the runFor method with their starting burst time as the run time and a system clock variable as the system clock. After they have ran, I performed some calculations and then pushed them into another priority queue. This second priority queue uses a comparator based on the process ID instead of the arrival time. Because we will run through all processes only once, it is easy to see that the removal from and the insertion to a priority queue of all processes have 2*O(n log n) time complexity. This plus some other constant work has total time complexity of O(n log n). Finally, to output all the data, I removed the processes from the finished queue one at a time and wrote the appropriate data to the output file. This had time complexity of O(n log n). (The same output method was used for all the schedulers.) The total FCFS algorithm has time complexity of O(n log n).

The SRTF scheduler was built on top of the work done for the FCFS scheduler. I added a third priority queue to represent an intermediate place where processes would be stored. When the system time passes a process's arrival time, the processes are moved from the arrival queue and into this "ready" queue. It is sorted by remaining burst time. This adds an additional level of complexity, but did not increase the time complexity of the algorithm. Every time a process gets interrupted (in my implementation it actually was only allowed to run until an interruption would have occurred), the process would be put back in the ready queue. This could happen n times. This only add a factor of O(n log n) to the run time. The total SRTF algorithm has time complexity of O(n log n).

The PS scheduler was built on top of the work done for the SRTF scheduler. It added the idea to keep track of each process's start time. This was to ensure that each running process could accurately run at the time the shares necessary for it to run came available. Although a bit more challenging to program, the time complexity did not change from FCFS scheduling algorithm. That being O(n log n).

All the time complexities discussed assumes that Java uses a heap implementation of the priority queue. This ensures that insertions and removals have O(log n) run time complexity.