PA02 Write Up

Name: Adam Caudle
ID: 011774063

Intro:

For this assignment, I was tasked with creating a simulation that would implement three different CPU Scheduling algorithms (First Come First Serve, Round Robin, and Shortest Job First). To do this we were given example scheduling files and would have to open the file, parse it for the correct information (while avoiding comments), run our simulations, and report the results. This report will detail the implementation strategy for the scheduling algorithms and how I incorporated tie-breaking rules.

Implementation:

1. For FCFS my implementation was relatively simple. I created a loop that would continue until every created process had been finished. During this, I would perform operations on three lists. One list for blocked processes, one for ready processes, and one for running processes. I additionally included a tracking node for moving through the lists during operations. To correctly implement my algorithm, I created a function called "fcfs_add_arrivals" that simply treated my ready list as a queue. This meant that all processing coming in first would then be run first. This was a clean and effective solution. To handle the three tie-breaking rules I just had the order of my code specifically executed such that ties would be caught in the correct order early on. After the blocked processes were added to ready, and any arrivals were added, I checked the first tiebreaker of CPU time. If a process was done, I ended it before any additional operations could be considered. If the tie wasn't with CPU time, within the same if-statement, a process would be blocked based on its CPU burst time. With two of the possible ties checked, the last check (possible preemption based on RR quantum) was executed next. This one applies specifically to RR, so it was considered last. After all the checks, the scheduler continues operations in the correct order until all processes are completed where it then prints the summary and statistics.

2. For RR my implementation was very similar to FCFS. Because RR follows many of the same rules with an added quantum value, it follows the same set of operations and checks with an extra condition of checking the quantum value based on a decrementing counter. This kept the same desired order of tie-breaking checks just with an added condition. Other checks and operations were kept identical within the implementation. As a whole, this implementation kept the same looping structure that looped through until all processes were finished, performed operations on my process lists based on the process data read from the file, checked quantum values, and performed operations specific to the RR method and handled all tie-breaking conditions in the correct order.

3. For SJF my implementation was identical to FCFS with a couple of notable differences. While the structure of the simulation code is the same, the functions for adding arrivals are different. As opposed to using a queue for FCFS, processes are added by comparing their needed run time against all others within the ready list. This ensures that the shortest job is added to the front of the list whenever it is added to the ready list. This maintained the correct order, while also allowing code reuse to ensure that the tie-breaking checks were in the correct order. This resulted in correct SJF execution with tested results lining up with those provided by the instructor. This implementation kept the same looping structure that looped through until all processes were finished, performed operations on my process lists based on the process data read from the file, inserted processes based on their CPU time needed (putting the shortest in first) and handled all tie-breaking conditions in the correct order.

Conclusion:
By reading the process file, correctly parsing it, and performing simulation operations on the process list based on the sorting algorithm, I was able to get identical results to the provided tests. This shows that my simulation correctly reads and delegates CPU time based on all three of the CPU scheduling algorithms. This assignment helped me learn a lot about how CPU process scheduling works and produce a final product that lines up with provided tests.