# CS433 Programming Assignment 3 Lab Report

Lucas Birkenstock and Connor Toro

## Problem Description

Our task in this assignment was to implement several CPU scheduling algorithms discussed in the course. These include the first come first serve, round robin, priority, and shortest job first algorithms. Implementing these requires understanding of previous concepts such as process control blocks.

## Program Design

Our program was very structured and organized. For each scheduling algorithm, there was a header, cpp file, and driver file. Each scheduling algorithm had very similar implementations of the print_results() function, but generally differed greatly simulate(), init(), and other helper functions. Each scheduling algorithm has a similar pattern: the init() function deals with initializing and sometimes sorting a process list as input. The print_results() function prints average statistics on the waiting time and turnaround time. The simulate() function, as the name would suggest, simulates the algorithm. It updates variables associated with the average statistics, and runs until all processes have completed running.

Starting with first come first serve, this algorithm was the most simple. The init() function simply copies the parameter over to the member variable, since that is the order in which the processes will execute. The simulate function begins by copying the processes over to a readyqueue (deque). While the readyqueue isn't empty, a loop pops off the front PCB from the readyqueue, updates statistical variables, and prints the statistics for that particular PCB. This effectively just iterates through the entire queue. The print_results() essentially just calculates the statistical averages and prints them out.

The priority scheduler was also relatively simple. It also copies the parameter process list into a member variable, but is different in that it is sorted. To sort the process list, we call std::sort with a simple, but custom compare function. This function, compare_pcb, simply returns whether the priority of one process is greater than another. The simulate() function essentially iterates through the sorted list and updates statistical variables. The implementation of print_results() is similar to FCFS.

The round robin algorithm is a little more complex. Its init() function also copies the input process list, but also maps the PCB name to its burst time in a separate data structure. In the simulate function, while there are still processes left to complete, it runs through branching statements in a loop. For each process in the list, if its burst time is 0, do nothing. If its burst time is less than or equal to the time quantum, update the statistical variables, and increment the loop variable. Otherwise, if the burst time is greater than the time quantum, update the burst time variables.

Lastly, the shortest job first algorithm. Its init function is similar to the priority scheduler, in that it copies the input process list and sorts it with a helper function, except this time, it is sorted based on burst time instead of priority. The simulate function simply iterates through the list, updates variables, and prints data for each individual process.

## System Implementation

All in all, this assignment was not too difficult to implement. The most difficult part was probably figuring out how init() and simulate() were intended to work together, along with setting up the .cpp files and figuring out how to run the test files properly. Some scheduling algorithms were more complex than others, but none were overly difficult.

## Results

Our code meets all requirements except for the assignment on gradescope, except for the extra credit. It will correctly use first come first serve, priority, round robin, and shortest job first CPU scheduling algorithms.

## Conclusion

To conclude, all assignment requirements and functionality were met, and served as a good learning experience for understanding how various CPU scheduling algorithms work. This time around, we decided not to go for the extra credit. Next time around, we may go for it if we think it would be necessary for our grade. We could have started working on this assignment earlier, but we both wanted to take time off during spring break to simply enjoy break.