Anthony Devito 11775022

Lab 2 Report

For lab 2 we were tasked with simulating a scheduler using C. The three simulations we were required to implement were, first come first serve (FCFS), Smallest job first (SJF), and Round Robin (RR). Each jobs had similar fundamentals, were you took a process, assigned it a status then placed it in a queue. In the queue you'd typically send the process through many different checks each cycle, you're mainly looking for when the process will be done running, what's the status, are there any processes running, what's the burst time, among other things.

The idea with these algorithms is to efficiently run process, so for example, if you were designing a computer operating system, the OS would run smoothly. Some of the simulations were better than others, but each had their own unique challenges with handling ties. What I mean by that is if a you have two processes (A & B), each with an arrival time at 0 seconds- how would you handle that?

For FCFS, the way my algorithm handles that is it arbitrary picks one of the two processes to run first. In our example, it would happen to pick A first because A has a lower process ID than B. So B would have to wait for it's turn. This keeps the algorithm efficient and consistent

For SJF, similarly to FCFS, assuming burst times, priority level, arrival time are all equal. It chooses the process with the lower ID to run first. As above, if A & B get placed into my SJF algorithm, A goes first, followed by B.

For RR, tie breaking is handled a little bit differently, notably using quantum to run each task to keep track of how long a task is running. This leads to each task running equal amount of time per n cycles. For example, if we have our tasks A & B & C, we would run the scheduler like this, ABCABCABC, leading to more fairness compared to the other two algorithms. Which one goes first is determined by their order in the ready queue.

By simulating the scheduling algorithms for FCFS, SJF, and RR, we were able to learn more about the operational dynamics of process management in operating systems. The way each algorithm handles ties differently, be it through quantum allocation or process ID prioritizing, emphasizes how difficult it is to create scheduling techniques that maximize CPU utilization, efficiency, and fairness. In addition to reiterating the algorithms' theoretical foundations, this lab offered a hands-on look at how these algorithms are used in actual operating system design.