Project 2

Mahfuz Uddin
5/4/2022
CSCI 340-OS
Dr. John Svadlenka

A write-up of my software design and an explanation of the issues I encountered and resolved.

In this project, I had to mimic different operating system process scheduling algorithms. This is a multithreaded POSIX threads program that has the following components to schedule and execute operations. It has 2 threads, on top of the main thread doing specific jobs. The dispatch thread gets processes from the ready queue, runs them, and then sends it to a FIFO queue, where it waits until the logger thread fetches them. The dispatch thread runs differently depending on the type of scheduling algorithm chosen at the run time between, First Come First Serve (FCFS), Shortest Job First(SJF), or Priority. For FCFS, the dispatcher chooses the process that entered the ready queue first and gives it time to run. For the SJF, the dispatcher chooses the process with the shortest burst time in the ready queue and gives it time to run, finally for the Priority, the dispatcher chooses the processes with the highest priority level, and gives it time to run. After the processes are given time to run, it is then sent to a FIFO queue. The logger thread fetches the processes from the FIFO queue in a first come first served manner and writes the attributes of the Processes' control block to an output file. The FIFO queue is a shared resource since there are multiple threads accessing the FIFO queue, in this instance, the dispatcher and the logger. We need to lock the queue when one thread is accessing it, so the other thread doesn't access it at the same time to prevent corruption of information. To make sure only one thread has access to the shared resources, I used mutex locks before the start of critical sections and released it after the critical section. A critical section is the segment of code where processes access shared resources, in this stance when we are using the send() and receive() functions.

There were a few problems encountered during this Project. This project required a lot of working with pointers, so before anything, I brushed up on pointers. When starting this project, I knew the requirements for this project but I didn't know where to begin or how to implement them, so I took a few hours to browse the internet, reading about PCBs and the different scheduling algorithms. At first, I thought the SJF and priority scheduling would be hard to implement, but it's just iterating over the ready queue and finding the PCB with the shortest burst time or highest priority, respectively. A dilemma I had was choosing between queues or linked lists to implement the ready queue and FIFO queue, but in the end, I went with a ready queue because of the ease of implementation, even if the time complexity of removing was Big-O of N compared to linked list where it's constant.

This project was definitely on the difficult side of the spectrum similar to Project 1. Over the span of a few days, and long hours I was able to make my program work. At the end of the day, I learned so much about threads and scheduling algorithms, learning to mimic the operating system has helped me better understand how it works from doing this project.