Joshua Wang
5/5/22

CS340 Project 2 Writeup

This project was to make two pthreads that would schedule/dispatch and log processes to an output file using 3 different scheduling methods, utilizing function pointer arrays for both integer operations and scheduling. I initially thought about making the schedule/dispatch process lock then unlock for each individual send operation to the FIFO, so the logger would be able to concurrently write to the output files between sends, but I went against it because having to do so many context switches would have a rather high overhead, especially when the FIFO would also have to do multiple locks and unlocks throughout the writing process. I ended up making the logger's lock and writing process dependent on the FIFO being filled so the pthreads end up running and locking in a sequential order, so each thread would also only have to lock and unlock once.

This actually ended up being a solution to a problem I had with run order, as I initially made both my pthreads lock their mutex instantly with no condition behind them. Though sometimes the program ran fine in cases where the scheduler/dispatcher ran first, there was also a chance the logger would run first, which would instantly cause a deadlock in which the logger had nothing to receive and the dispatcher could not access the FIFO to send anything in the first place.

In the case of pthread creation, one of the issues I had was needing to pass in multiple values to the schedule/dispatcher even though it can only take in one parameter for its 4th argument. I ended up having to make a struct with multiple fields to be able to pass it into my pthread, and this struct was a rather core part of the parsing process, as I needed to know what my command line arguments were, and I needed to pass in my file input string to be parsed.

As for the parsing process, since each attribute of a process was split by either commas or newlines, I could use strtok() with commas or newlines as delimiters to get each attribute of a process. After creating a parsing process, and filling the ready queue with these attributes, the main next issue was scheduling based on the ready queue. For FCFS, since we parse in first order, the function does nothing and the ready queue will already be properly scheduled. For SJF and PRIORITY I used qsort to automatically number the ready queue to prioritize either burst or priority number. With scheduling done, I used my send function to put it onto my FIFO, which is just a global struct that stores 15 processes, and tracks the rear and increments with each send/decrements with each receive so more processes can be sent.

The logger was one of the easier parts of the project to implement, all I had to do was make sure it would write when the FIFO was full, and then write out 15 lines for each process. I suppose one "issue" was that my program was not as modular as it could be, since I specifically made the program run for a source file of 15 processes, since each FIFO was specified to only hold 15 processes and each source file had 15. However, a remedy for this issue would just be tracking the lines of each file, and replace these fixed values with however many processes were read instead.