Karim Rhoualem
Student #26603157

Due: Sunday, November 29, 2020
Section: FK-X

Programming Assignment #3 – Lab Report

High-Level Description of the Code

The purpose of this assignment is to create a process scheduler that combines and implements the Shortest Remaining Time First (SRTF) and Round Robin (RR) scheduling algorithms in order to read a list of processes from an input file, process them, and then write the corresponding execution steps to an output file. Threads are used to conduct the context switching between processes.

Prior to reaching the main entry point of the program, the Reader.java file must be addressed. In this class, a BufferedReader object is created to sequentially parse the lines in *input*.txt. Each line in the input file corresponds to one process. Each of those processes is loaded into a Queue object for later processing. The entry point of the program is through Simulation.java. Here, the Reader object is invoked on the input file, and the Queue is loaded with the processes. At that point, several new threads equal to the number of processes in the Queue are created and subsequently started. Each running thread is an invocation of the RRScheduler.java class. The RRScheduler has a static constructor that initializes several static variables used to keep track of all information related to the processes. Most notably, the thread number, quantum size, waiting time dictionary, process completion time dictionary, process completion time dictionary, and semaphore objects are some of the many tools used within the class.

When a thread's .run() method triggers, StartProcess() is called, which required that each thread wait on the semaphore to gain access to the critical section. This critical section begins by finding the arrived process in the Queue that has the shortest remaining time (SRT). Then, it logs/writes that it has started/resumed to console/file, updates the processes remaining execution time and the current program running time after it has simulated execution, and then it finally logs/writes that it has paused or completed execution. When the process is being updated, a time quantum q is deducted from the process' remaining time. This time quantum is a constant percentage of the remaining execution time of the process. When the processes have all terminated and the Queue is empty, the total waiting times are calculated for each process. This is calculated as: $Waiting\ Time = Turnaround\ Time - Arrival\ Time - Burst\ Time$. At that point, the final information about the waiting times is also logged/written to console/file and the program completely terminates.

Discussion

As a first iteration of the program, simulating the RR scheduler was easy. All that was required was that the threads be synchronized via a single semaphore, and within each critical section, the next process in the Queue would be removed, executed, updated, and then placed at the back of the Queue.

As a second iteration of the program, the SRTF criteria was implemented. Rather than removing the next process in the Queue at the beginning of each critical section, some logic was performed to find the next shortest process from the remaining processes. While doing so, the criteria that the longest of two processes with the same shortest execution time remaining be chosen to execute next, was also met. While looping through the remaining processes and searching for the shortest one, a search continues so long as a process' remaining time is less than or equal to the minimum time found. Once that logic completes, the process with the shortest remaining time is removed from the Queue and fed to the remainder of the critical section to be formally executed and processed using the RR scheduler.

The simulation is not perfectly accurate. In addition to having to select an arbitrary minimum time quantum, an arbitrary minimum remaining execution time cut-off also had to be selected. If one was not selected, the algorithm would run infinitely given that the remaining execution times would continuously be subdivided (however in reality, this would eventually end due to lack of space in memory to keep floating points). Due to these round-off errors, processes are completing their full execution times in periods that are somewhat shorter than they should be. For example, when time quantum of 10% is used and the minimum cut-off execution time of 0.1 units is used, Process 3 from the sample input file only completes about 75% of its execution time before terminating. As such, it ends up displaying a negative execution time. This is an extreme case of inaccuracy within the simulation, but it works relatively fine when other more appropriate numbers are selected. This issue would be difficult to eliminate without having more researched and simulation-backed cut-off values to work with.

Some advantages to using this hybrid method over the pure RR technique is that there is a good balance of fairness and speed. Given that the time quantum is constant for all processes regardless of their sizes, the RR algorithm may create long wait times when larger processes execute. This is inefficient for the shorter and possibly higher-priority processes. By integrated SRTF, shorter process terminate more quickly, while also allowing longer processes a chance to execute periodically without experience starvation.