Programming Assignment 3 Report

Andres Hinojosa

Submitted Files:

- **pcb.h:** Contains the code for the PCB objects and their variables (added variables).
- **scheduler.h:** The base scheduler class which will be overridden.
- **scheduler_fcfs.h and .cpp:** Subclass for FCFS scheduling algorithm header and implementation.
- **scheduler_sjf.h and .cpp:** Subclass for SJF scheduling algorithm header and implementation.
- **scheduler_priority.h and .cpp:** Subclass for priority scheduling algorithm header and implementation.
- **scheduler_rr.h and .cpp:** Subclass for RR scheduling algorithm header and implementation.
- **scheduler_priority_rr.h and .cpp:** Subclass for priority-based RR scheduling algorithm header and implementation.

How to compile and run the program:

- To compile the program, use command "make" followed by the scheduling algorithm.
- To run the program, use command: ./ followed by the scheduling algorithm, input file, and quantum time if it includes RR.

Results for programs:

- All my scheduling algorithm implementations were able to successfully complete the included test and Gradescope test without any errors.
-

Features implemented:

- **FCFS**: I first worked on the first come first serve algorithm as it was the simplest. I implemented this schedule algorithm by including a list of the PCBs. Because I transferred each in the order they arrived, I simply went from index 0, to the end and simulated the algorithm by counting the time units that have passed by. Each object arrived at 0, waited at the time they started, and had a turn-around time equal to the time they finished.
- **SJF**: The next scheduling algorithm I worked on was the shortest job first. I implemented this algorithm by also including a list which I passed each PCB to. This time, I iterated through the list, comparing its burst time to the last to save the index of the shortest job that has been found. From there, I simulated the algorithm by marking that PCB as finished, adding to the passed time, and repeating until all PCBs have been marked as visited. Each PCB arrived at 0, waited until the time they started, and had a turn-around time equal to when they finished.
- **Priority**: After that, I worked on the priority scheduling algorithm. The process was very straightforward as the same implementation I used for SJF can be re-used with modifications. I re-used the algorithm but made it so that it is looking for the PCB with the smallest value in its priority variable. Everything else was the same with being marked and repeated. Each PCB

arrived at 0, waited until the time they started, and had a turn-around time equal to when they finished.

- RR: The final required scheduling algorithm that I implemented was round robin and required some thinking. My implementation used a list of the PCBs that have been passed by and is like the FCFS in the way it looks at each PCB in order. The difference is that each process can only run for the same amount of time as the quantum time or less. I did this by only subtracting from the burst time and repeating until each PCB had a burst time of 0. Each PCB arrived at 0, waited for the sum of each time they started subtracted by the number of times they ran before their last run multiplied by the quantum time, and had a turn-around time equal to when they finished.
- Priority_RR: The final scheduling algorithm that I implemented was the priority-based round robin and was the most difficult. My implementation was a combination of RR and Priority. It is priority on the outside, and round robin on the inside. Since prioritization comes first, we look at the highest and simulate the process by allowing it to run for the same amount of time as the quantum time or less. The process repeats until all PCBs of the current priority have a burst time of zero, then it moves on to the next priority by decrementing a counter until all PCBs are finished. Each PCB arrived at 0, waited for the sum of each time they started subtracted by the number of times they ran before their last run multiplied by the quantum time, and had a turn-around time equal to when they finished.

Design and implementation choices:

This programming assignment was the first one that I had started knowing how each algorithm works, so it was not as difficult for me. I designed each implementation file similarly as they all used the same constructor, destructor, results function, except for the simulate function. I first worked on the examples on paper to make sure I knew what was going on. From there, I coded my thought process which was especially helpful for the round robin algorithms. I was able to understand how to calculate the turn-around times based on a formula.

Lessons Learned:

From this assignment, I learned that fully understanding the assignment before starting will benefit me. I did not have much trouble except for an error in thinking that a smaller priority value meant a higher priority. This was also the first time I used the class discussion which helped me to further understand and finish my program quickly.

References:

The reference I used were the lecture slides for each algorithm to make sure I understood the Gantt charts.

Future improvements:

To improve my program, I would optimize the time complexity for the algorithms that include priority scheduling. They are both a for loop inside a while loop, so if there were many processes, it would take a while. Other than that, I would remove redundant code and find ways to compact it more.