Ben Foltz-Miranda

Dr. Zhang

CS 433-02

29 March 2024

# Assignment 3 Program Report

## Problem Description

The problem that I am solving is being able to schedule PCBs with different scheduling algorithms. These scheduling algorithms are, first-come, first-served (FCFS), shortest-job-first (SJF), priority, and round-robin (RR) scheduling. The point of being able to schedule these different algorithms is that we can emulate how processes would run on a computer, when running multiple different processes, they are going to be run with one of these algorithms.

## Program Design

Since this program has a few different designs since each algorithm requires different things, I will describe the overall design of each individual algorithm and how I implemented it in the code. First, there is the FCFS algorithm, this one was an overall very simple algorithm to implement. I created a queue of PCBs that would hold the processes that would be run, a variable to hold the total number of PCBs, a variable to hold the total turnaround time, a variable to hold the total wait time, and two vectors, one to hold the turnaround time and the other to hold the waiting time of each PCB so it could be printed when running the processes. The FCFS init function loops through the PCB vector “process\_list” and pushes the elements onto the queue and sets the “original\_size” variable, which holds the number of processes, to the size of the queue. The print\_results() function loops through each element in the queue and prints a which process is running and what the turn-around time and waiting time is for each element by pulling it from the vectors that stored that information and then printing the total wait time and average wait time for running all the processes. In the simulate() function, I loop through the queue until it is empty and on each loop I pop off the front element and then print out which process is running and for how many time units it is running for. After, the turnaround time and waiting time are calculated and added to the vectors used to store them, then the time is updated to represent how much time has passed while running the process and the total turnaround time and waiting time are updated for use in the print\_results() function. For the SJF algorithm, it is mostly the same as FCFS, but instead of using a normal queue to store the PCBs, I created a priority\_queue that would store PCBs as a vector, and would use a custom comparison that would check the burst time of the two PCBs and return either true or false depending on if PCB p1 had a larger burst time than PCB p2. I also had to create a map to hold the turnaround and waiting times for each process that was ran since it would be easier to keep a hold of which turnaround and waiting time went with each PCB. The priority algorithm was exactly the same as SJF except instead of comparing the PCBs by their burst time, they were compared by their priority and then ordered in the priority\_queue. The round-robin algorithm did not require a priority\_queue so I used a normal queue like before but kept the maps for turnaround and waiting time. The main difference was in the simulate() function where I had to create another map that would hold the original burst time of each PCB so that the turnaround and waiting time could be calculated. It also required that we check if the burst time of the current PCB is greater than the time quantum or not, if it was greater, than it would update the PCBs burst time and then print out how long it was run for and the add the PCB back to the queue, if it was not greater than the quantum time, then it would run it for the remaining burst time and then update the turnaround and waiting time. The last algorithm, priority round robin is a similar combination of how the round robin algorithm and priority algorithm work. I used all the variables from the priority algorithm, but changed how the simulate() function worked to be able to run processes with the same priority in a round robin schedule. I made a map in the simulate() function that would store a queue of PCBs with the same priority. After, I loop through the priority queue, popping off the top PCB and then adding the original burst time of each PCB to the map, and then pushing the top PCB to the map of PCB queues. Then I loop through every entry in the map of queues and retrieve the PCB queue and check if the size of the queue is greater than 1, if it is then it will loop through that queue until it’s empty in a round robin schedule, if it is not then it will run the single PCB fully.

## System Implementation

I did run into some problems during my implementation of the scheduling algorithms. One of the issues that I ran into was figuring out how to create the simulate() function for the round-robin scheduler and the priority round-robin scheduler. With the normal round-robin scheduler, I struggled with figuring out how to keep track of the proper turnaround time and waiting time for each process. What I ended up doing was creating a map that held the name of the PCB as the key and either the turnaround time or the waiting time as the value that way, when it came to printing out the results, I was able to just loop through the queue and pull the values associated with each PCB. With the priority round-robin scheduler, I struggled with how to determine which PCBs needed to be run in a round-robin schedule and which ones could be run normally and how to make sure that it would do it like that. I had to use another queue to store PCBs with the same priority so that it could run those until they were all fully run and then go back to the main queue to run the rest of the processes.

## Results

I believe that all the features that were required are included in my submission. I think that something that I would like to improve on would be going back to the first few algorithms that I implemented and make changes that would make more sense, since I was completing each algorithm one at a time and taking the main ideas from the previous algorithm and improving on it as I continued onto the next algorithm. All of my scheduling algorithms passed the tests on gradescope, including the priority round-robin test.

## Conclusion

I was able to solve the intend problem successfully. The program is able to successfully schedule processes in each of the scheduling algorithms, FCFS, SJF, Priority, RR, and Priority RR. A lesson that I’ve learned from this assignment is that I should create more generalized functions so that I can reuse code instead of having to copy code that is similar and changing it a bit. I liked that this project had each algorithm as its own individual project that could be done without having to implement every part of it, it made it a lot easier to work one each part a little at a time since I could worry about just implementing on algorithm and once that one worked I could move onto the next one.