Programming Assignment 3 Report

Submitted Files:

- **pcb.h**: A class representing a single task. Add your names to the file header. No other changes are needed for this file.
- **scheduler.h**: A base class for different schedulers. It defines several pure virtual functions, init, print_results and simulate, which are implemented in the subclasses.
- **scheduler_fcfs.h and .cpp**: A subclass of Scheduler implements the FCFS scheduling algorithm. The init function initializes the scheduler with an array of incoming processes. The simulate function simulates the scheduling of processes in an FCFS ready queue. The print_results function prints out the statistics, including the turn-around and waiting time of each process and the average turn-around and waiting time.
- **driver_fcfs.cpp**: A driver program tests the SchedulerFCFS class. It calls the member functions init, print_results and simulate.
- **driver_sjf.cpp, scheduler_sjf.h and .cpp**: Files for SJF scheduling.
- **driver_priority.cpp, scheduler_priority.h and .cpp**: Files for priority scheduling.
- **driver_rr.cpp, scheduler_rr.h and .cpp**: Files for RR scheduling.
- **driver_priority_rr.cpp, scheduler_priority_rr.h and .cpp**: Files for priority-based RR scheduling (extra credits).
- **Makefile**: Allows for automation for having to compile and link object files of the above files using the “make” command and creates executable ELF files: fcfs, sjf, priority, rr. Also creates object files (.o) for all the source files

How to compile and Run the Program:

To compile the program, use command: make

To run the program, use command: ./fcfs schedule.txt; ./sjf schedule.txt; ./priority schedule.txt; ./rr schedule.txt 10

Results/Test cases:

Our program successfully completed the effort we put forward on the various test cases and our functions are functioning to the extent of what we know. The test cases are displayed in the Gradescope autograder.

Features Implemented (by class):

pcb_stat: *a struct that contains instance variables relating to id, waiting time and turnaround time.*

SchedulerSJF, SchedulerFCFS: *Contains vector<pcb_stat> stats, and queue<PCB> process_queue, also contains current_time, sum_waiting_time, and sum_turn_around_time.*

SchedulerRR: *Contains what is provided in SchedulerSJF or SchedulerFCFS, but considers the time_quantum input field.*

Additional Features:

- *Utilizes std::queue container from the STL within the C++ containers library*
- *Utilizes std::vector container from the STL within the C++ sequence containers library*

Design and Implementation choices:

Our design and implementation choices were derived from what we learned in class, which includes the slides performing the various features and operations of the many scheduling algorithms we worked on in class. We decided to work using the vector and queue STL containers, as it helped make things easier, compared to using a doubly linked list be associated with a vector<stat> and the queue<PCB>, which would have made it more complicated than it really needed to be. We first approached prioritizing on the FCFS scheduling algorithm, successfully providing results from the simulate function. We eventually made our way to the SJF and provided what we know into our implementations. We made our way to the Priority test, and we eventually caught up to an obstacle regarding the priority test, however it eventually turned out to be an easy fix from a problem with the algorithm. We also attempted the RR scheduling algorithm.

Overall understanding and learning outcomes:

We learned various operations for the usages and tests regarding the scheduling algorithms. This helped us understand processor scheduling. It provided a fundamental deal of knowledge into how we could abstract these ideas and we may even go about implementing these algorithms into our own personal projects.

References:

We used the lecture slides and ZyBooks's sections as references to familiarize ourselves when working on the project. We also took familiarity with looking upon the lecture examples and homework quizzes when we were doing these scheduling algorithms.

Future improvements:

Future improvements we consider would be most likely working more on the attempted extra credit segment. This will help us better understand on providing our implementation regarding the RR scheduling algorithm.