Montoya, Saiah 029177653
Alferos, Peyton 029011474

Project 4 Report

VIDEO WITH EXECUTION AND OUTPUTS:
https://youtu.be/KySwmsKVi9g

Group Work Breakdown

Peyton Alferos:

- Implemented and debugged FCFS class and Priority class
- Implemented “Task Finished” print statement in RR class
- Recorded output video (sent to Saiah for upload)
- Created readme

Saiah Montoya

- Implemented and debugged RR class
- Added comments to all classes
- Wrote majority of Project report
- Uploaded output video

In this project we Implemented the following CPU scheduling algorithms, First-Come-First-Serve (FCFC), Round-Robin (RR), and Priority queue. We create a list within each algorithm to hold all of the tasks entered in the CPU schedule, we called this List<task>.

In the FCFS implementation, we have two methods. Schedule() iterates over each task in the queue and processes them in the exact order they appear. For each task in the queue, the method executes the task on the CPU for its entire required burst time, the total run time of the task. After a task has been run, the method prints a message to the console indicating that the task is finished. In PickNextTask(), the system selects and returns the next task to be processed. If the queue is empty, it returns null, and if it is not empty, it removes and returns the first task in the list. This ensures that processes entered first will be completed first.

In the Priority queue implementation, after tasks are entered in list, they are ordered in descending order to ensure that the queue executes with the highest priority. We use two override methods, schedule and picknexttask. Schedule processes each task in the queue in order of their priority, and iterates through the sortes list, running each task on the CPU for its entire burst time, and when the task completes, a message prints showing the task has finished. In the picknexttask, the system checks to see if the queue is empty, and if it’s not, it will remove and return the first task in the list, which has the highest priority.

In the RoundRobin queue implementation, we added a private integer Quantum, to represent the quantum value in which processes will be ran for. In the schedule() method, the task queue is iterated over, and each task is scheduled. It dequeues a task, runs it for either its remaining burst time or the quantum time (whichever is smaller), and then checks if the task is completed. If a task's burst time exceeds the quantum, its burst time is updated, and the task is re-queued at the end of the queue. This process ensures each task gets an equal share of CPU time in a round-robin fashion. A message is printed when a task's quantum time is finished.