Cory Hershman

Scheduler Post-Mortem

This post-mortem will discuss the steps that were taken to complete the Scheduler project. First, the requirements were analyzed and listed. Next, classes and methods were proposed. The proposed classes were Process, Scheduler, and main. The Process class would have 4 variables: arrival time, burst time, waiting time, and turnaround time, with get and set methods. However, it was later found that more variables were necessary, manipulating the arrival and burst times was cumbersome, error-prone, and unnecessary. The following variables were added: remaining time, starting time, and finishing time. The Scheduler class would be an array list of processes with 3 methods: FCFS (First Come First Serve), SJF (Shortest Job First), and RR (Round Robin). Each of these methods would output an array list of strings to store the processing schedule. The main class had no design or pre-planning and was designed after the Process and Scheduler classes. The idea was to keep the implementation details of the main class separate from the other classes. Next, algorithms were created for the FCFS, SJF, and RR methods. The original pseudo-coded algorithms were accurate, with the aforementioned exception that some extra variables were needed. Finally, the main class was designed. The main class would have 2 methods: createTable, and toGanttChart. After scheduling, the createTable method would create a table out of the resulting variables of the processes. The toGanttChart method would create a gantt chart out of the schedules that were output from the scheduling algorithms. Due to the extra methods in the main class besides the main method, it made sense to rename the class as it became a reusable class for making gantt charts rather than just a main method. The main class was renamed as SchedulerSimulation. The scheduling algorithms run in pseudo-real-time, so the algorithms may be analogous to scheduling algorithms used in real-time schedulers. The downside to implementing the scheduling algorithms this way, is that the processes have to be entered in order of arrival time, just as a real-time algorithm would. Errors will occur if the processes are not entered in order of arrival time. After the project was completed, a design oversight was found. The scheduling algorithms search after each time unit through the scheduler list for the next process to run. It would be more efficient to keep a separate stack for adding processes that have arrived and popping off processes that have finished. This would reduce the run-time of the algorithm and simplify some sections of the code as opposed to the current algorithm which performs the same operations as a stack but without shortening the list of processes to search. A stack implementation would also be closer to a real-time implementation. This change would be fairly simple but would

require time to modify the rest of the code in the algorithms. For now, the change has not yet been made. The most important lesson learned in this project was the ability to make decisions based on pseudo-real-time operation.