**PROJECT 3: WRITE-UP**

1. **SUMMARY**

This project demonstrates a discrete event simulation to test various scheduling algorithms. The program takes in two arguments – the first is a scheduling algorithm to use and the second is the name of the file that specifies the processes to run. The program simulates the execution of the processes on a single processor and after the simulation it prints out statistics.

1. **IMPLEMENTATION**

The project implementation is using a Linked List to separate the jobs of the program. After the OS executes it takes in the first argument which is the scheduling algorithm: either First Come First Serve, Round Robin, Shortest Process Next, Shortest Remaining Time, or Highest Response Ration Next. The second argument specifies a process to run. The processes are divided into events which have a duration of time in either the CPU or IO event. The CPU runs and the IO blocks.

Time is then tracked with a single integer variable. It starts at zero and increments representing the passing of time. When a process is selected it is dispatched and the next event is a CPU event. Time counter is incremented by the duration of the event and the next event is handled. This keeps getting repeated until the next event is an IO event, or the algorithm tells it to do something else. IO events block the process and put it in a queue for the duration. The program should keep track of all time and statistics and then print them to the screen after all of the processes are finished.

Arrival Time is the first number of the process argument. Service Time the of time the processes spent in possession of the CPU (with the sum being the durations of all CPU events). Start Time is the time the first gains control of the CPU. Finish Time is the time when the last event of the process is complete. Turnaround Time is the duration between the start time and finish time. Normalized Turnaround Time is Turnaround Time divided by Service Time. And the Average Response Time is the response time it takes the process to gain control of the CPU after encountering an IO event unless it is the last event.

**III. EXPERIENCE**

🡪 POSITIVE 🡨

I feel like I had a good understanding of this project at first and I probably could’ve consolidated some of my classes a little bit more. I took most of my previous knowledge from a project I did at Brookhaven with the Linked List and used inheritance and polymorphism to combine all into a single program. I also feel strongly that I was able to properly infuse each of the five scheduling algorithms; although quantum in the Round Robin took forever for me to get right, I think I succeeded in that as well. Though my architecture is a little too lengthy and of course printing the correct output (which I’ll explain later) was not what was asked as a deliverable, I feel I accomplished a lot with this one.

🡪 NEGATIVE 🡨

I came across 2 major obstacles that I could not overcome, and they are still bothering me probably still as you’re reading this. First, the service\_Given Boolean was something that seemed to escape my understanding. Why was it necessary? What was it doing? And, where to implement it? I ended up entering it into the 3 correct algorithms, but I don’t think it’s working correctly. Without understanding those 3 questions, I was unable to solve that issue. And finally, what took me days to find out I could not accomplish, was printing out the proper output. I was struggling understanding your example output pdf at first. Then I realized that averaging out those times was impossible for me because I couldn’t figure out how to tell when an event ended and when one began. After exhausting myself and without that knowledge I was unable to succeed with the output. But despite that, overall, I feel I came out a better programmer and have a better understanding of the entire concept of scheduling and processes.