Name: Daniel Young

Course: CS302 Operating Systems

Due Date: 03/19/2014

Assignment: CPU Scheduling Team Project

This project was created to meet the requirements established by the Team Project handout provided in class. It consists of three code files and a pair of input files. The program aims to simulate two different scheduling types: First-Come, First-Serviced (FCFS); and Round-Robin (RR). Time is measured in “units,” and CPU percent utilization and total time required to run a particular number of processes are reported after running a simulation.

The first methods developed for this program centered on user input and main method loops in the “Sim” class file. The goal was to provide a user with the options to play/replay simulations, enter options for those simulations, and ask for help if needed. These methods were designed with the belief that users would be running the program without arguments and not from a command line. Considering this, command line argument support was added in later, which required minor code updates with additional “if/else” and “for” checks. This was the creator’s first use of command line arguments in a Java program, so research was required. The only major issue with argument input comes with user-preferred filenames, as any wrong command input will be assumed to be a filename. This detail will not be an issue at all if the user correctly enters arguments.

The next step in developing the program involved setting up the “Process\_” class file. Knowing that processes would need to be simulated in the program, a class was required to keep each process’s variables such as waiting time, turnover time, arrival time, ID, etc. Also, methods to set and retrieve those values would be needed for proper encapsulation, which, in the end, was ruled relatively unimportant for this simulation program anyway. Turnover and waiting time would require recalculation upon process completion; otherwise, their values would be unreliable. Constructors were needed to properly create Process\_ objects when provided values from the input file mentioned in the handout. These values were process number, arrival time, and burst time. So, while each process has a large number of instance variables associated with it, only three were taken from the input file and fed to the constructor. No major issues were experienced in this file creation.

Once the main method loop and Process\_ class were created, a third class (CPU\_Simulation) was required to actually run simulation calculations, load the text input file into the program, populate a set of Process\_ objects from that file, and provide text output to the user after the simulations were run. Considerable thought was put into how to implement the Process\_ objects in the simulation, and it was decided that an array of the Process\_ objects would be used to keep track of them all. The array was deemed static to the CPU\_Simulation class, as no more than one array should be created at any time for the program’s execution. The developer believed that, if a single input file was to be used to compare two different scheduling algorithms, then that same file could be stepped through twice to repopulate the same array if it was needed. No array was to be worked on simultaneously by two algorithms, either.

Another method would be required to print out information of processes in the array after the simulations were complete. This method was completed before the simulations, as the requirements were clear on what output was required. A parameter was passed to determine if the output would be detailed or standard. CPU Utilization and total time required were to always be provided in the printout, regardless of the parameter passed. There were no issues in designing this method besides a few minor syntax errors that were easily corrected.

In the FCFS simulation, the array of Process\_ objects was stepped through and executed in order of process ID, taking very little else into consideration (other than process arrival time and order). A queue could have been used for this simulation but was deemed completely unnecessary considering the simplicity of FCFS scheduling requirements. A final adjustment was made at the end of the scheduling algorithm that would advance the clock if processes had not yet arrived; this adjustment, however, was deemed unneeded and slightly inaccurate. It was left in to prevent infinite looping of the algorithm.

In the RR simulation, the array of Process\_ objects was checked for arrival time and a Queue was created. The Queue only kept the order of process execution. Otherwise, all Process\_ objects kept their completed time and burst time variables to themselves. These variables were checked by the RR scheduling loops to make sure each Process\_ object was only worked on for a unit of time equal to a “QUANTUM”, which was declared constant for the CPU\_Simulation class. This QUANTUM was equivalent to 10 clock units, but could be changed by the user before compilation if need be. It is at the instructor’s discretion to test this change if need be.

Upon realization that the user should be able to enter his or her own input file for the program, the developer decided to implement a new CPU\_Simulation class variable known as “file”, which could be changed via a “set” method. The set method would receive its input from the Sim program command line arguments, which meant a String variable would be passed by the Sim class to the CPU\_Simulation class. The developer was disappointed that this minor detail was missed at first, as it required a small overhaul of the two classes. However, the changes were minor overall.

The end product was a program that could be executed with or without command line arguments. If the user starts the program without arguments, the program provides the user to change options on the fly (with the exception of the input file name, which defaults to input\_default.txt without command line arguments). If the user includes command line arguments upon opening the program, the arguments must be “-d,” “-a” (followed by either FCFS or RR), and/or a filename. If other arguments are entered, or if these arguments are entered incorrectly, the program will throw some sort of error.

It is felt that, while the program structure could be more modular and separated, the majority of the code flowed well together in the same class. If more objects were required for a simulation, additional classes could have been used and would have been separate from the CPU\_Simulation class, but this was not the case. Also, while a third scheduling simulation (SJF) could have been programmed, the developer worked on the project alone and felt there would have been inadequate time to “polish” what could have been the final product.

It is my opinion that, from these two example scheduling simulations, RR is the better choice of algorithm. While FCFS seemed to reduce context switch time, RR seemed to be more able to average out turnaround times. It might be user preference, but for the developer, having many processes complete around the same time would be preferred to each process completing in order. However, the context switch time would have to be considerably smaller than the average CPU burst times of the processes for this to be a truly valuable scheduling algorithm.

When the context switch time is reduced to only one time unit, the scheduling algorithms become much more efficient and use much more of the CPU’s time. When the context switch time is changed to ten, the CPU’s time is used considerably less efficiently, and processes spend a considerable time waiting to be completed. This, of course, is based on an average CPU burst time of only 20 time units, so having half of a process burst time equal to the context switch time would indeed make the switch inefficient.

This program was compiled using DrJava. The exact version name was drjava-20130901-r5756, build time of 20130901-2137. So long as the input\_default.txt, Process\_.java, CPU\_Simulation.java, and Sim.java files are all in the same directory, the program should compile and function correctly. Should the user find any issues with the program, please contact the developer at [Daniel\_P\_Young@yahoo.com](mailto:Daniel_P_Young@yahoo.com) for a prompt response.