Kevin Do
SID 10447610
CS 143A Programming Assignment Report
Professor Venkatasubramanian
10 June 2015

First Come First Served Scheduling

For this algorithm, I created a class called Process to hold all information about the data coming from the input file. This made managing and organizing the information much easier. To start off the algorithm, I first read in the data from an input file. I then organized it from earliest arrival time to latest arrival time. I used selection sort ($O(n^2)$ time complexity) for organizing the processes. I figured since the testing data set isn't large, selection sort would be adequate. If the testing set was much larger, I would have used a much quicker algorithm such as merge sort or quick sort. I approached the problem by simulating a computer's cycle. By using a while loop, I can count the CPU's "cycles" in every iteration. For every iteration, I iterated through the array of processes to see which processes were waiting according to their arrival time and the number of past iterations. While also incrementing the processes' wait times, I also incremented the current running process' executing time. When the executing time of the current process is equal to its burst time read in by the file, we know the process is done, so a boolean called "finished" is set to true to mark the process has completed. The while loop continues until all the processes in the array are finished. Before outputting, I used selection sort once again to sort the processes by their process ID's. This way the processes are organized nicely for output. Overall, the FCFS algorithm I implemented is $O(n^2)$ complexity due to the selection sorts and the netted for loop within the while loop.

Shortest Remaining Time First Scheduling

For SRTF, I reused the Process class in the FCFS algorithm. Many methods of the Process class used in FCFS was also used in SRTF, but with a few more added functions such as timeRemaining(). Like the FCFS algorithm, I used selection sort to organize the processes by arrival time after reading from the input file. Also similar to the FCFS algorithm, I simulated a computer's cycle by using the same while loop. This time, I used a helper function to give me the index of the next process to execute. This function looked through the array of processes and finds the least remaining burst time within the processes that have arrived. Again, after every loop iteration, I incremented the waiting processes' waiting times. The current process' executing time is incremented after every iteration until the remaining burst time is zero. Once it is zero, the process has completed. The while loop continues until all processes in the array are finished. Once again, like in FCFS, selection sort is used to organize the processes by process ID's before outputting. In many ways the SRTF algorithm is very similar to the FCFS algorithm. The overall complexity of SRTF is also O(n^2) due to the selection sort and the nested for loop within the while loop.

General Thoughts

Both algorithms could have been slightly more efficient by implementing merge or quick sort rather than an inefficient selection sort. Also, since I simulated a "computer's cycle," the algorithms tend to have a higher complexity. If I were to do raw calculations rather than iterating each tick of the cycle, the algorithm would probably be much faster than it is right now.