# An analysis of CPU Scheduling Algorithms

For assignment 5, we have been assigned the task of analyzing two CPU scheduling algorithms. To do this, we have to complete the following six tasks: select two CPU scheduling algorithms, implement a CPU scheduling simulation, test both algorithms in the simulation, compare and contrast the scheduling criteria, compare and contrast the performance, and provide a situation where each algorithm performs optimally. The two CPU scheduling algorithms I chose were First-Come First-Serve (non-preemptive) and Round Robin (preemptive). Non-preemptive means the process keeps the CPU until it either (1) switches from the running state to the waiting state or (2) terminates. Preemptive means the process keeps the CPU until it (1) switches from the running state to the waiting state, (2) switches from the running state to the ready state, (3) switches from the waiting state to the ready state, or (4) terminates. FCFS is a simple algorithm: the first process which requests the CPU is processed first. The average wait time for FCFS is long due to the convoy effect. The convoy effect is the slowing of the Operating System due to a process taking a long time to terminate. RR is similar to FCFS, but with preemption added. The average waiting time for this policy is long. I built the Simulation and tested the two scheduling algorithms. The requirements of the assignment state that we test the CPU utilization, throughput, and turnaround time. The CPU utilization is how busy the CPU can get when executing the source program. Throughput is the number of processes per second (pps). Turnaround time is how long each algorithm takes to run on the same Job Queue.

I designed the simulator in C using XCode on a MacBook Pro 2019. The code utilized Mac OS Specific kernel databases to get information on the CPU. The program initializes a Job Queue, enqueues a bunch of random jobs. Each job is at most 5 milliseconds apart from each other in arrival time and have a minimum burst time of 2 milliseconds and a maximum burst time of 42 milliseconds. The program will then run the First-Come First-Serve algorithm on the Job Queue, and then run the Round Robin algorithm on the Job Queue next. Finally, the Queue is destroyed. Refer to the source code for more information.

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | First-Come First-Serve | Round Robin | Difference |
| CPU usage | 10.508146% | 10.508185% | 0.000039% |
| Throughput | 68,750 pps | 423,077 pps | 354,326 pps |
| Turnaround | 0.000160 s | 0.000026 s | 0.000134 s |

RR CPU usage is 0.000039% faster than FCFS, RR can execute about 354,326 more processes, and RR is 0.000134 seconds faster.

Building a simulation was somewhat difficult, mostly because I was unable to test preemptive vs non-preemptive situations. The performance will end up being mostly focused on the algorithms I chose to represent each of the schedulers. For this program, I did a sequential search and wait for FCFS, but for RR I utilized the total time for the Job Queue and just waited for that amount of time. So, the results answer to some extent the questions posed, but I think I would gain more information if I were to either use the POSIX Real-Time Scheduling SCHED\_FIFO and SCHED\_RR to implement the algorithms or to build a scheduler for a custom OS. Regarding which algorithm might work better than the other, let’s consider a time-sharing system. FCFS would perform terribly due to the Operating System locking on each process. What if one user had a really long process? Then all the other users would get locked out. RR on the other hand is suited perfectly to time-sharing systems. It will allow each user to process a set amount each iteration, which at certain speeds would appear to be consistently processing everything they wanted.