**Scheduling Algorithms Analysis**

**Performance:**

We ran a few different sets of processes, with varying cycle sets to get a good feel for the average run times of the various scheduling algorithms.

1. The first set of algorithms we ran generated processes with the number of cycles randomly generated to have a mean of 6000, and a standard deviation of 1000.

Order (least to greatest): SRT, SJF, FIFO, RR, LRT

1. The second set of algorithms we ran generated processes with the number of cycles randomly generated to have a mean of 6000, and a standard deviation of 4000.

Order (least to greatest): SRT, SJF, FIFO, RR, LRT

1. The third set of algorithms we ran generated processes with the number of cycles randomly generated to have a mean of 3000, and a standard deviation of 6000.

Order (least to greatest): SRT, SJF, FIFO, RR, LRT

1. The fourth and last set of algorithms we ran generated processes with the number of cycles all set to 3000.

Order (least to greatest): SRT/SJF/FIFO, RR, LRT

1. Average of the 4 sets.

Order (least to greatest): SRT, SJF, FIFO, RR, LRT

In every case, the SRT algorithm resulted in the shortest average waiting time, followed very closely by SJF. FIFO resulted in the next shortest average waiting time. RR resulted with the second longest average waiting time in every case due to the recurring context switching penalties. LRT had the longest average waiting time due to the recurring context switching when process cycles were near-level resulting in repeated context switches.

**Differences:**

**RR:**

The round robin algorithm could have probably used a longer quantum for our processes. Since the round robin algorithm is a preemptive algorithm, the relatively short quantum (50 cycles) was much less than any of the process cycles, created a heavy penalty for switching between processes since the quantum was so small compared to the average number of cycles per process. The round robin algorithm

**SRT:**

The shortest remaining time algorithm finished faster than all the other algorithms consistently. Although it was also a preemptive algorithm and had context switch penalties, it was able to lower the average waiting time more than the non-preemptive version of the same algorithm, shortest job first. This algorithm, like shortest job first assumes we know the length of the cycles in the process. It appears that the cost of the context switches was worth the cost to lower the average waiting time for the other processes. The shortest remaining time algorithm was one of the algorithms that required the processes to be sorted, in order to get the process with the least number of remaining cycles. While the average wait time was lower for the processes, the constant sorting/finding might have increased the time which was not calculated in our tests.

**SJF:**

The shortest job first algorithm consistently got very good results (second least average waiting time). This algorithm is non-preemptive, and assumes we know the length of the cycles in the process which isn’t always the case. The shortest job first algorithm was one of the algorithms that required the processes to be sorted, in order to get the process with the least number of remaining cycles. While the average wait time was lower for the processes, the constant sorting/finding might have increased the time which was not calculated in our tests.

**FIFO:**

The first-in-first-out algorithm is simply a queue. The first process that arrives is the first process that gets to execute. This algorithm is non-preemptive and simply executes the processes in the order they arrive. First-in-first-out, although it didn’t have great average waiting times, didn’t create any unnecessary context switches which kept the average waiting times below that of the round robin and longest remaining time algorithms.

**LRT:**

The longest remaining time algorithm (if there was such an algorithm) created many repeating context switches resulting in very high penalty costs and resulted in the longest average waiting times. Once all the existing processes have the same number of cycles remaining, longest remaining time takes the context penalties of switching between all the processes continually. The longest remaining time algorithm also assumes we know the number of cycles for each process. The longest remaining time algorithm was one of the algorithms that required the processes to be sorted in order to get the process with the greatest number of remaining cycles. While the average wait time was lower for the processes, the constant sorting/finding might have increased the time which was not calculated in our tests.

**Overall:**

With the current quantum, it seems first-in-first-out might be the most efficient use of CPU time considering the other two better performing algorithms required some sort/find to get the processes with the least number of cycles. Even if the data structures holding the processes were using a linked list where new entries could be inserted in order there would be lookups while inserting each new process.

Perhaps with a longer quantum, the round robin might have been able to perform quite a bit better resulting in a lower average waiting time. The problem with the current quantum is that it’s so small that not enough work is getting done on individual processes before they are switched out. When the quantum is too long it results in first-in-first-out, and by comparison first-in-first-out resulted in much lower waiting times.

The longest remaining time algorithm is completely useless. All the processes end up on the same number of cycles remaining, and then a small amount of work is done on all of the processes, and the number of cycles gradually decreases across the board, but with the penalty of the context switches, the cost of this algorithm is much higher than the other algorithms.

The interesting test case we ran was where all the processes had the same number of cycles. Shortest remaining time, shortest job first and first-in-first-out all had the same results (as shortest remaining time and shortest job first default to first-in-first-out when there is a tie), but round robin and longest remaining time both still had significantly higher average waiting times. This shows the potential flaws of those algorithms in their current conditions.