Scheduling Algorithms Analysis

Daniel Biwer
James Stephens

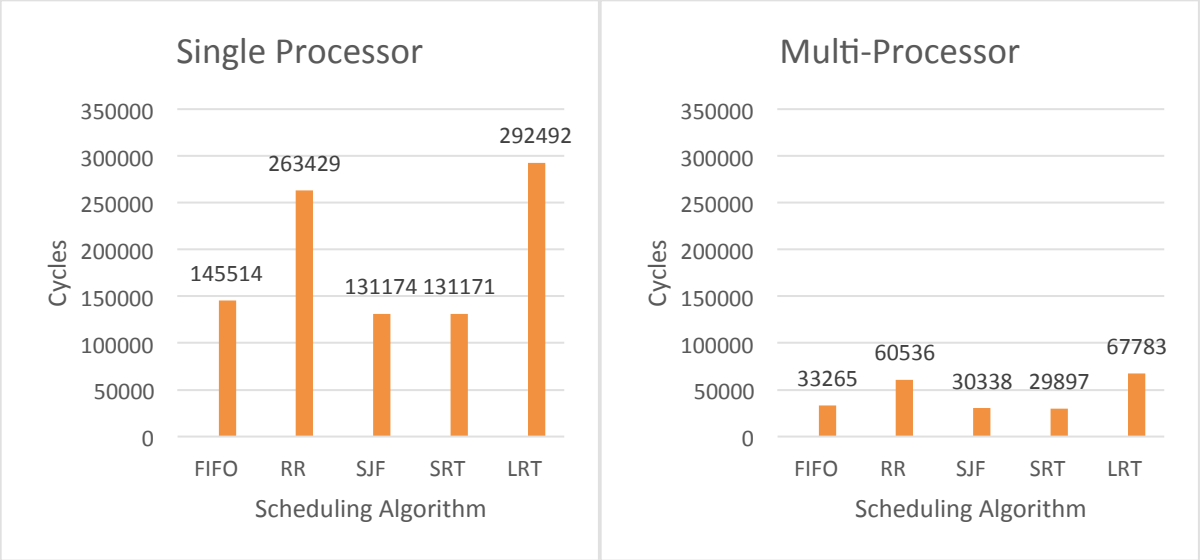
CSCE 4600.003

Scheduling Algorithms Analysis

Performance:

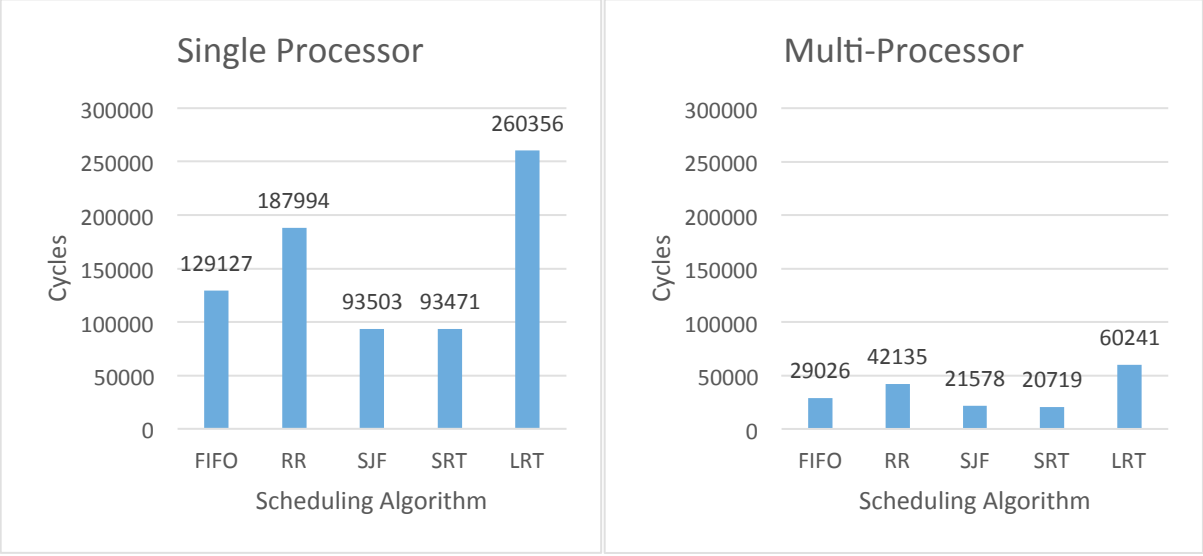
We ran a few different sets of processes, with varying cycle sets to get a good feel for the average run times of the various scheduling algorithms.

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.



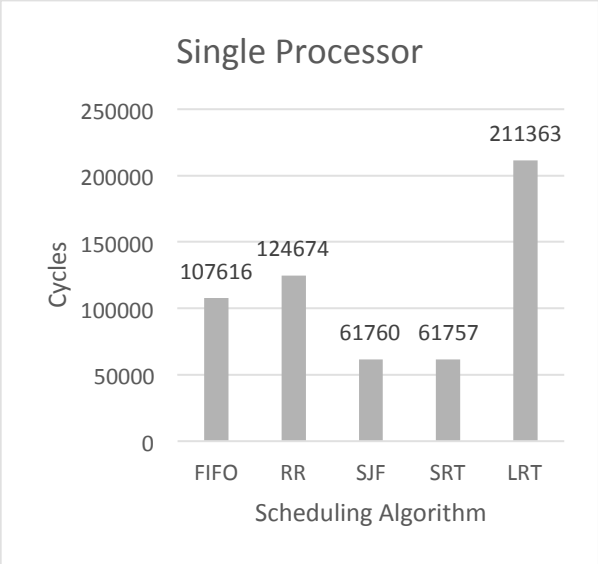
Order (least to greatest): SRT, SJF, FIFO, RR, LRT

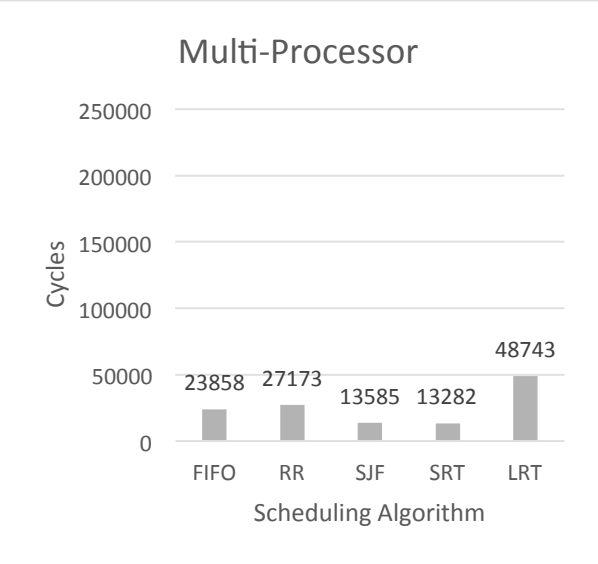
2. The second set of algorithms we ran generated processes with the number of cycles randomly generated to have a mean of 6000, and a standard deviation of 4000.



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

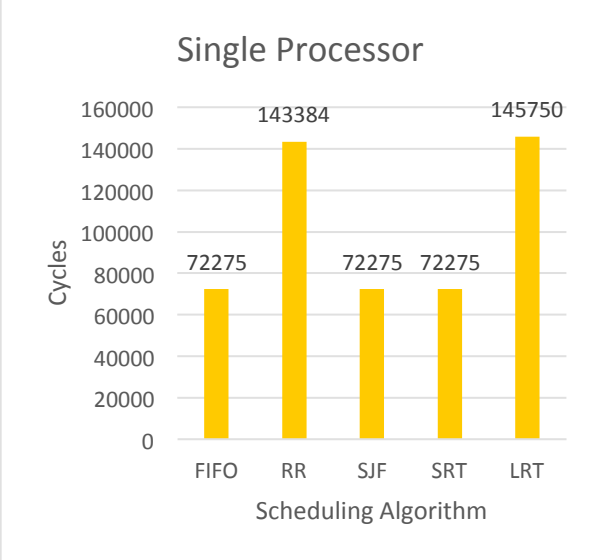
3. The third set of algorithms we ran generated processes with the number of cycles randomly generated to have a mean of 3000, and a standard deviation of 6000.

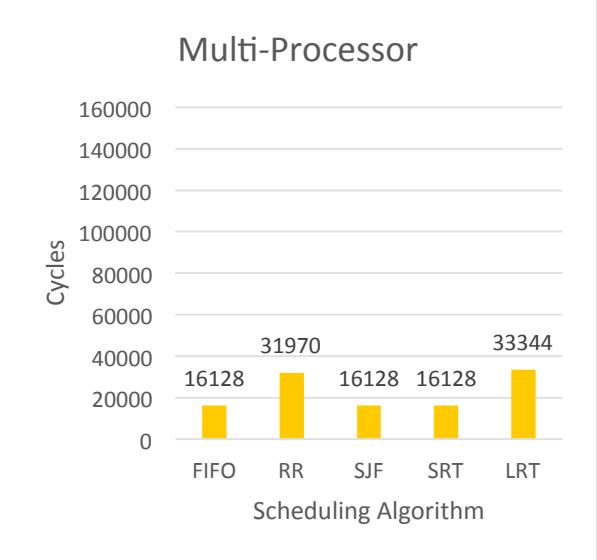




Order (least to greatest): SRT, SJF, FIFO, RR, LRT

4. 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

5. 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.

Conclusion:

With the current quantum, the shortest job first and shortest remaining time algorithms seem to make the best use of CPU time. These two algorithms consistently produced the same wait times for the same sets of processes. For this case, the preemptive SRT algorithm would probably be the most useful. It would not change the wait times in any significant way, but very small processes would be removed from the wait queue very quickly. The decision to use SRT over SJF would be determined by the kind of jobs being performed by the system and personal preference.

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 extremely inefficient. Eventually, all of the processes approach the same number of remaining cycles, and the processes begin to be switched on every cycle until their completion. This means that the context switch penalty is incurred on every cycle. These penalties add up very quickly, as can be seen by the resulting graphs from the simulations.

Another factor to consider, which was not accounted for in the simulations, is the cost of performing a sort algorithm in the scheduling processes that would require it. The SJF, SRT, and LRT algorithms all require a way to select the process with the least or greatest cycles. SJF and SRT required the least amount of sort calls, only requiring the processes to be sorted when a new process was added. The LRT scheduling algorithm required a sort to be performed on every cycle. The use of sorting could potentially increase the context switch required to pick the next process.

If sorting time were implemented in this simulation, the first-in-first-out algorithm would make a case for being the most efficient scheduling algorithm. FIFO did not require any sorting or searching, and had the fewest context switch penalties.

The most interesting test case that we ran was when all of 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 function the same way as 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 the RR and LRT algorithms in their current conditions.