Project 2: System Calls and Schedulers

Our group attempted to measure a fairness metric (and have a developed version of calculating it stashed locally), but with the throughput assumptions that all processes would have to be “wanting more time on the CPU”, it didn’t seem reasonable because all of the executables we tested go to sleep for I/O operations so often. We would find that many processes may have 0 ticks on the CPU, but most of the time the reason a process isn't on the CPU is just because its blocked, not because the scheduler leaves them waiting. We decided to take a look at turnaround and response time instead.

We used a command defined in “schedulertest.c” to test 20 executables running on a single CPU (changed this in the makefile for testing). We figured it’d be easier to see the efficiency of the scheduler for one CPU so there’s no actual parallelism going on, and all 20 processes would be trying to use the same processor. The executables are:

5 copies of stressfs

5 copies of find

5 copies of cat README | uniq

5 copies of wc README

The function prints out the max, min, and average turnaround and response time of the run.

	RR			Stride		
	Max	Min	Avg	Max	Min	Avg
Turnaround	298	14	175	261	12	140
Response	2	1	1	6	0	1

	RR			Lottery		
	Max	Min	Avg	Max	Min	Avg
Turnaround	298	14	175	290	10	181
Response	2	1	1	5	1	1

	Stride			Lottery		
	Max	Min	Avg	Max	Min	Avg
Turnaround	261	12	140	290	10	181
Response	6	0	1	5	1	1

In our results, we found that when we look at the Round Robin (RR), Stride, and Lottery scheduling, they all showed pretty similar turnaround and response times. In our setup, neither the Lottery nor Stride scheduler made use of a priority mechanism when it came to stride length or number of tickets, they simply used random values. We figured this would be why they operated in a way that was so like Round Robin; to get the full benefits of those two schedulers, you'd want to use these values strategically.

What we did notice was that the Stride scheduler had a better turnaround time, suggesting it might be a bit more efficient compared to both Round Robin and Lottery, allowing some processes to make use of multiple ticks at a time to catch up to the others. On the other hand, the Lottery scheduler performed slightly worse, likely because its random approach can sometimes lead to less-than-ideal scheduling choices.

While we initially aimed to measure fairness metrics, we found that a bit tricky. Still, the turnaround and response time data we gathered gave us some useful insights into how these scheduling algorithms stack up against each other. Round Robin's "perfect" fairness is mostly reflected in the response time metric. The RR scheduler has the least variance when it comes to response time; the max response time was 2, while the others had maxes of 5 and 6. this shows that no one process experienced significantly slower response time than another, as RR is expected to behave.