Report: Jinda Cui (cuij2)
Project done with Jordan Charest

OS Project 1 - Analysis

1. Best Algorithms

Of the three simulated algorithms, which algorithm is the best algorithm?

FCFS is the most basic algorithm. It may work very well for certain kinds of tasks where it can be seen as a greedy algorithm. But in our case, we should focus on comparing the other two algorithms which are more sophisticated.

SRT and RR all have advantages and disadvantages. It's hard to say which one is the best. SRT can be fast but facing starvation problem. RR emphases fairness, no starvation problem but has larger overhead/average waiting time and requires tuning on the time slice to get the best performance. In our test cases, SRT appears to perform the best.

Which algorithm is the best for CPU-bound processes?

RR is the best for CPU-bound processes. Not only because RR is fairer and more responsive to processes, but also for SRT prefer shorter bursts while CPU-bound processes tend to have longer bursts. This is observed in test 1, in which process B and D are CPU-bound. In SRT B started using the CPU at 180ms and D at 870ms. However, in RR the two numbers are 92ms and 444ms.

I also made a case which shows more apparent difference:

A|0|300|5|30

B|0|66|5|30

C|5|70|5|30

D|7|75|5|30

E|15|300|5|30

F|20|40|7|5

G|25|22|12|5

Here two CPU-bound processes are A and E. With RR time slice set at 75ms.

In SRT A started using CPU at 1556ms, although it arrived at 0ms. But in RR it started using the CPU at 4ms. Similar behavior can be observed for process E. Also, although RR is in general slower than SRT, it gives more number of bursts for these tasks distributed all over the timeline.

Which algorithm is the best for I/O-bound processes?

SRT is the best for I/O-bound processes. The reason is I/O-bound processes tend to have shorter CPU burst time, which is preferred by the SRT algorithm. Once finished CPU burst, these processes can wait for I/O without counting waiting time.

In test 1 we can observe this behavior: in the base case (FCFS) we have 3 bursts for C (I/O-bound) at last, but in SRT we have only 2, which means C complete a burst in prior. RR also has 3 bursts of C at last. A similar tendency can be observed on A too.

2. For the RR algorithm, how does changing rr_add from END to BEGINNING change your results?

Intuitively, changing rr_add from END to BEGINNING should affect little on CPU-bound processes. Since they'll stay on the ready queue after preemptions, the average turnover time and average waiting time will be minorly affected. However, for IO-bound processes, if BEGINNING is specified, they can jump in front of the ready queue, fast finish their bursts and go back to IO blocked status. This potentially can significantly reduce their waiting time and turnaround time. Sum up the delay in CPU-bound processes and the speed-up in IO-bound processes, we may see performance increment depends on the ratio of the two classes (choice of t_slice may affect this too).

These behaviors can be observed from the test cases we were given. In p1-input00.txt (test 0), the two processes are all CPU-bound. For END, the average wait time is **100.00** ms and the average turnaround time is **214.00** ms. For BEGINNING, the average wait time is **102.00** ms and the average turnaround time is **216**.00 ms. We see **slight increase** on both measures which conform with our analysis.

In test 1 we have CPU-bound and IO-Bound processes mixed together. For End, the average wait time is **315.08** ms and the average turnaround time is **746.00** ms. For BEGINNING, the average wait time is **288.31** ms and the average turnaround time is **718.00** ms. Here we observed apparent speed-up in the BEGINNING case. This is as predicted in our analysis.

3. Again for the RR algorithm, can you tune t_slice such that you attain the shortest average turnaround time or shortest average wait time for

the given sample test input files? Explain what happens when you use different values of t_slice.

A heuristic for an optimal RR performance is 80% of CPU burst times should be less than the time slice. If too much time is assigned the RR degrades to FCFS. If too small a time slice it will be very slow.

For example, in test case 04, if we assign a time slice of 104ms, around 80% of the processes will be within the time slice, the average wait time is 319.71ms, and the average turnaround time is 430.88ms. This is better than 80ms time slice. if we assign a time slice of 120 ms > any CPU burst times, the RR average wait time and turnaround time are 332.71ms and 443.21ms respectively, which are the same with FCFS. If we define the time slice as 20ms, then there are too many context switches such that it becomes extremely slow. The two values become 527.71ms and 672.54.

In some other cases, we may have multiple local minimums, means the RR performance goes up and down as we tune the time slice. However, I could never make RR average times better than SJF.