Michael Quach

Operating Systems

Fall 2017

Table of Contents

Table of Contents 1

Introduction 2

Flow Charts 3

Results 4

Discussion 5

Samples of Execution 7

Simulation Results 10

Source Code 11

Introduction

Similar to humans, a computer is only capable of performing one task at a time. In order to choose which task to perform next, a computer's operating system has a scheduler tell it what to do. A scheduler can employ one of several different algorithms that prioritizes different things in the computer's to-do list; however, in practice, schedulers often mesh a number of different algorithms rather than using a single, pure algorithm.

This experiment attempts to simulate a few specific scheduling algorithms, which were decided upon by my instructor, Dr. Furht. These algorithms are "first come, first served" (FCFS), "shortest job first" (SJF), and "multi-level feedback queue" (MLFQ), in addition to the multi-level feedback queue using "round robin" (RR) for two of its priority layers. For the sake of simplicity, the first two algorithm simulations will include not preemption. The experiment simulates these algorithms in the aforementioned order, re-initializing the necessary variables after each scheduler simulation is complete.

The FCFS algorithm is very simple: the process that comes into the ready queue first gets the CPU first. Once it finishes its task, it moves to an input/output queue (IO), and then re-enters the ready queue at the end. This continues until all processes have completed all of their tasks.

SJF is somewhat similar to FCFS; however, instead of simply serving the first process, SJF serves the process with the shortest task. After that, it moves to IO as usual and then back to the ready queue when the IO is done.

MLFQ is essentially several layers of schedulers with each layer employing a specific scheduling algorithm. In this experiment, the first layer (priority) employs the Round Robin (RR) algorithm. RR is also similar to FCFS, but instead of always allowing a process to finish its task, the process is cut short after a specified time period, called Tq (for time quantum).

If the process finishes its task before its period ends, then everything is fine. If it doesn't, it is downgraded to the second priority, which also uses RR in this case. Processes in the first priority always run before the second priority. If a second-priority process still takes longer than the Tq allows, it is downgraded to the third priority, which is FCFS in this case. This experiment does not allow processes to be upgraded, so once one slips down one priority, it is not allowed to go back up one.

Discussion

As expected, FCFS arguably performs the worst among the simulated algorithms. With the lowest CPU utilization (82.02%) and the highest average waiting time (285.875) and average turnaround time (691.5), FCFS takes the longest to allow a process to finish and leaves the CPU idle for a significant duration. It remains significantly better than SJF with regards to average response time (36.25), but this is not a determining factor, and it is still notably worse than MLFQ's average response time. It is worth noting that its average waiting time is not only the worst among the three, but it is almost a staggering 50% higher than either of the other algorithms.

SJF fares a tad better than FCFS, but has noticeable issues, especially when its performance in the middle of the simulation is inspected. SJF boasts the lowest average waiting time (168.875) and turnaround time (574.5) among the three algorithms, as well as a notably higher CPU utilization than FCFS. It's no wonder, too, considering the algorithm allows processes with quick bursts, such as P1, P3, and P5, to execute their bursts quickly without much waiting for any of them.

However, the average response time (81.875) is more than twice as high as either FCFS or MLFQ, and a look at the individual processes' statistics shows some worrisome trends. In particular, P7 has more than twice the waiting time of any other process (485) and the highest turnaround time (855), which is a significant difference to the other end of the spectrum, P3's turnaround time of 288. A look at the individual bursts of the processes makes it obvious as to why this occurs: P7 not only starts with one of the longest CPU bursts among the processes, but *all* of its CPU bursts are individually longer than any other process's bursts. As a result of this combined with the SJF algorithm, P7 must wait until it is the *only* process in the ready queue in order to get started on its very first burst; P7's sky-high response time of 476 is evidence of this. In addition, it has to wait for this condition of solitude in order to perform any burst at all, since it will have lower priority than literally any other process, no matter which specific burst it is on. As a result, it can be concluded that, although SJF is an acceptable algorithm when each process has similar burst times, one outlier can skew the statistics greatly and be left to starve on its slightly longer burst while every other process jumps between ready and I/O constantly.

Finally, MLFQ (this specific MLFQ, at least) reaches a nice middle ground between the extremes of FCFS and SJF. Its average waiting time (201.875) and turnaround time (607.5) are at a good medium, but it boasts the lowest average response time (18.875) and the highest CPU utilization (92.375%). Although it has extremes in the low-end of individual waiting times, with P1's waiting time of 25 and P5's 22, it does not have any extremes on the high-end, like SJF had with P7. It does, however, have consistently higher individual waiting times compared to SJF. In addition, despite its higher average turnaround time compared to SJF, the individual times on the processes are not significantly different, although comparing the times of one process between the two algorithms does show some glaring differences.

Ultimately, for the purposes of this experiment, MLFQ appears to be a favorable algorithm. It does not have extreme edge cases of starved processes, like SJF has, but it is more efficient in distributing the CPU than FCFS. And, of course, boasting the highest CPU utilization among the three means less time is wasted on waiting for I/O to finish.

However, in general, MLFQ is noteworthy for its flexibility. It is effectively an algorithm that chooses different algorithms to use. The MLFQ setup in this experiment left longer bursts sitting in the FCFS-priority queue for periods of time where the shorter bursts were preemptively claiming the CPU. A number of different setups could be used in an attempt to alleviate the issues that this MLFQ setup had, such as notably lower average waiting and turnaround times.

That being said, it is worth also mentioning that, since MLFQ simply uses other algorithms, it means that FCFS and SJF (and, indeed, most other algorithms) are by no means worthless just because of their flaws. For example, instead of using RR for the first priority queue in MLFQ, SJF could be used instead, with a RR or FCFS algorithm in the second priority, with preemption disabled. That's not to say that this setup is perfect, but it helps to show the flexibility in scheduling and, ultimately, the difficulty in creating an excellent scheduler, simply by virtue of how many possibilities there are.

Samples of execution

FCFS:

PCB for P1

Total time elapsed: 4 Response time: 0

Waiting time: 0 Turnaround: 4

Moving to IO.

Ready q: deque([P2[18], P3[6], P4[17], P5[5], P6[10], P7[21], P8[11]])

IO q: deque([P1[24]])

PCB for P2

Total time elapsed: 22 Response time: 4

Waiting time: 4 Turnaround: 22

Moving to IO.

Ready q: deque([P3[6], P4[17], P5[5], P6[10], P7[21], P8[11]])

IO q: deque([P1[6], P2[31]])

PCB for P1

Total time elapsed: 28 Response time: 0

Waiting time: 0 Turnaround: 28

Moving to Ready.

PCB for P3

Total time elapsed: 28 Response time: 22

Waiting time: 22 Turnaround: 28

Moving to IO.

Ready q: deque([P4[17], P5[5], P6[10], P7[21], P8[11], P1[5]])

IO q: deque([P2[25], P3[18]])

PCB for P4

Total time elapsed: 45 Response time: 28

Waiting time: 28 Turnaround: 45

Moving to IO.

Ready q: deque([P5[5], P6[10], P7[21], P8[11], P1[5]])

IO q: deque([P2[8], P3[1], P4[42]])

PCB for P3

Total time elapsed: 46 Response time: 22

Waiting time: 22 Turnaround: 46

Moving to Ready.

PCB for P5

Total time elapsed: 50 Response time: 45

Waiting time: 45 Turnaround: 50

Moving to IO.

Ready q: deque([P6[10], P7[21], P8[11], P1[5], P3[4]])

IO q: deque([P2[3], P4[37], P5[81]])

SJF:

PCB for P1

Total time elapsed: 4 Response time: 0

Waiting time: 0 Turnaround: 4

Moving to IO.

Ready q: deque([P2[18], P3[6], P4[17], P5[5], P6[10], P7[21], P8[11]])

IO q: deque([P1[24]])

PCB for P5

Total time elapsed: 9 Response time: 4

Waiting time: 4 Turnaround: 9

Moving to IO.

Ready q: deque([P2[18], P3[6], P4[17], P6[10], P7[21], P8[11]])

IO q: deque([P1[19], P5[81]])

PCB for P3

Total time elapsed: 15 Response time: 9

Waiting time: 9 Turnaround: 15

Moving to IO.

Ready q: deque([P2[18], P4[17], P6[10], P7[21], P8[11]])

IO q: deque([P1[13], P5[75], P3[18]])

PCB for P6

Total time elapsed: 25 Response time: 15

Waiting time: 15 Turnaround: 25

Moving to IO.

Ready q: deque([P2[18], P4[17], P7[21], P8[11]])

IO q: deque([P1[3], P5[65], P3[8], P6[35]])

MLFQ:

In Priority 1.

Q1: deque([P1[4], P2[18], P3[6], P4[17], P5[5], P6[10], P7[21], P8[11]]) Q1 IO: deque([])

Q2: deque([]) Q2 IO: deque([])

Q3: deque([]) Q3 IO: deque([])

In Priority 1.

Q1: deque([P1[3], P2[18], P3[6], P4[17], P5[5], P6[10], P7[21], P8[11]]) Q1 IO: deque([])

Q2: deque([]) Q2 IO: deque([])

Q3: deque([]) Q3 IO: deque([])

In Priority 1.

Q1: deque([P1[2], P2[18], P3[6], P4[17], P5[5], P6[10], P7[21], P8[11]]) Q1 IO: deque([])

Q2: deque([]) Q2 IO: deque([])

Q3: deque([]) Q3 IO: deque([])

In Priority 1.

Q1: deque([P1[1], P2[18], P3[6], P4[17], P5[5], P6[10], P7[21], P8[11]]) Q1 IO: deque([])

Q2: deque([]) Q2 IO: deque([])

Q3: deque([]) Q3 IO: deque([])

PCB for P1

Total time elapsed: 4 Response time: 0

Waiting time: 0 Turnaround: 4

Moving to IO.

In Priority 1.

Q1: deque([P2[18], P3[6], P4[17], P5[5], P6[10], P7[21], P8[11]]) Q1 IO: deque([P1[24]])

Q2: deque([]) Q2 IO: deque([])

Q3: deque([]) Q3 IO: deque([])

In Priority 1.

Q1: deque([P2[17], P3[6], P4[17], P5[5], P6[10], P7[21], P8[11]]) Q1 IO: deque([P1[23]])

Q2: deque([]) Q2 IO: deque([])

Q3: deque([]) Q3 IO: deque([])

In Priority 1.

Q1: deque([P2[16], P3[6], P4[17], P5[5], P6[10], P7[21], P8[11]]) Q1 IO: deque([P1[22]])

Q2: deque([]) Q2 IO: deque([])

Q3: deque([]) Q3 IO: deque([])

In Priority 1.

Q1: deque([P2[15], P3[6], P4[17], P5[5], P6[10], P7[21], P8[11]]) Q1 IO: deque([P1[21]])

Q2: deque([]) Q2 IO: deque([])

Q3: deque([]) Q3 IO: deque([])

In Priority 1.

Q1: deque([P2[14], P3[6], P4[17], P5[5], P6[10], P7[21], P8[11]]) Q1 IO: deque([P1[20]])

Q2: deque([]) Q2 IO: deque([])

Q3: deque([]) Q3 IO: deque([])

In Priority 1.

Q1: deque([P2[13], P3[6], P4[17], P5[5], P6[10], P7[21], P8[11]]) Q1 IO: deque([P1[19]])

Q2: deque([]) Q2 IO: deque([])

Q3: deque([]) Q3 IO: deque([])

P2 is being downgraded to priority 2.

Simulation Results

----------Simulation complete----------

Time elapsed:901

Times: Waiting Turnaround Response

P1: 318 670 0

P2: 283 756 4

P3: 341 563 22

P4: 209 746 28

P5: 272 901 45

P6: 313 610 50

P7: 229 599 60

P8: 322 687 81

Averages: 285.875 691.5 36.25

CPU Utilization: %82.01997780244173

----------Simulation complete----------

Time elapsed:855

Times: Waiting Turnaround Response

P1: 42 394 0

P2: 239 712 81

P3: 66 288 9

P4: 234 771 45

P5: 69 698 4

P6: 108 405 15

P7: 485 855 476

P8: 108 473 25

Averages: 168.875 574.5 81.875

CPU Utilization: %86.4327485380117

----------Simulation complete----------

Time elapsed:800

Times: Waiting Turnaround Response

P1: 25 377 0

P2: 313 786 4

P3: 74 296 10

P4: 263 800 16

P5: 22 651 22

P6: 310 607 27

P7: 269 639 33

P8: 339 704 39

Averages: 201.875 607.5 18.875

CPU Utilization: %92.375