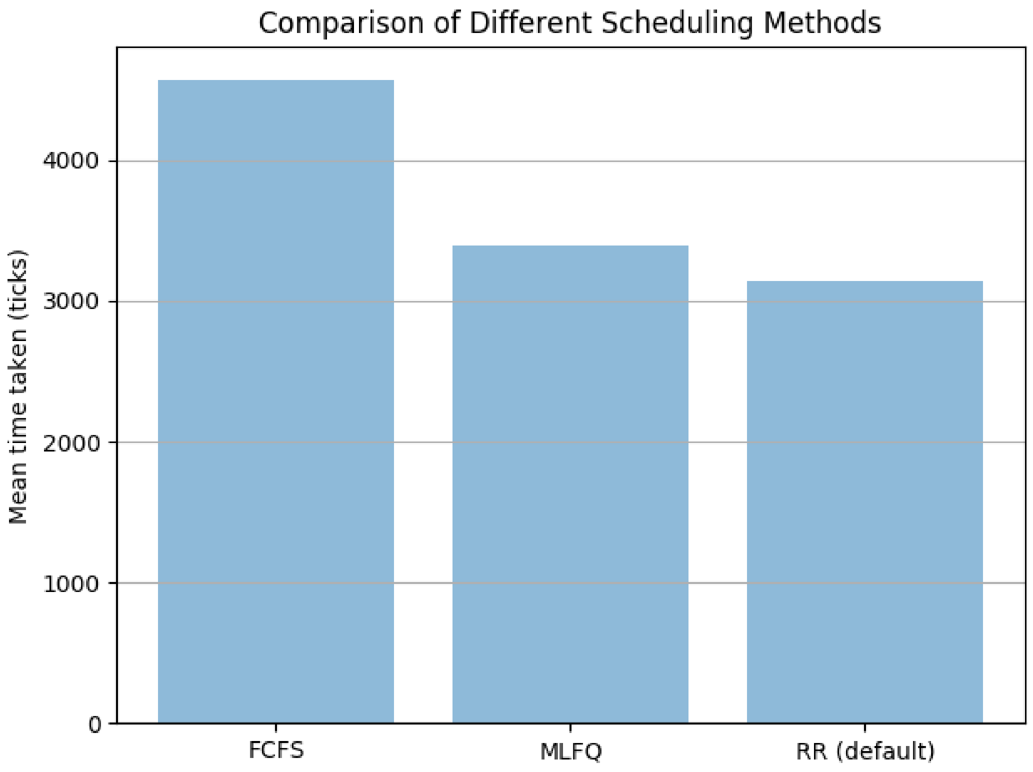
Report

Benchmarking performance of scheduling algorithms:



Commands to run:

- *> make clean; make qemu-nox CPU=1 SCHEDULER=<type>*
- *$ bench*

The code in bench.c is used to test all scheduling algorithms. It forks 10 times and the parent waits for the 10 children processes to terminate. The child processes spend different sleeping and utilizing CPU so overall, there is a good blend of both I/O and CPU. Note: PBS is shown later.

Conclusions:

- FCFS gives worst performance as it has the highest endtime – creation time (wait time + run time)
- RR kind of gives the best performance.
- MLFQ is slightly slower than RR while PBS is faster than FCFS although slower than MLFQ.

Ranking:

1. RR (default)
2. MLFQ
3. PBS
4. FCFS

Inference:

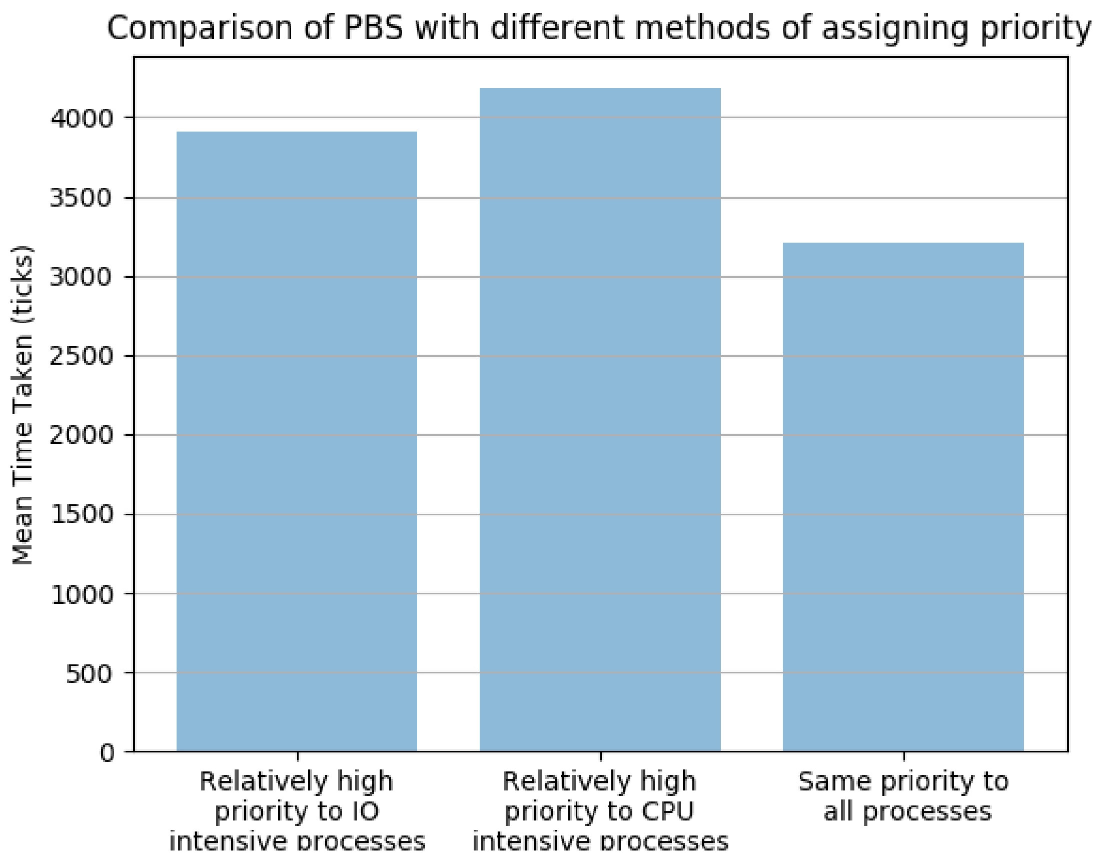
1. Round Robin gives the best performance as all processes (regardless of CPU or IO) were given same amount of time and yielded after their time quanta was over. This was I/O intensive processes didn't have to wait for CPU intensive processes, hence preventing the Convoy Effect.
2. First Come First Serve gave the worst performance since CPU intensive programs came first so were served first. The I/O intensive programs had to wait needlessly despite have short burst times.
3. In Multi-level Feedback Queue system, I/O intensive processes went to sleep (yield) frequently so they were retained in higher priority queues. CPU intensive processes took more 'ticks' and hence were constantly shifted down to lower priority queues.

PS: Processes can exploit this by pre-empting (interrupting) themselves just before the time-slice of the queue intentionally so that they remain in the highest priority queue although they may be very CPU intensive.

Priority Based Scheduling

Ten child processes had been created by forking. The ith (child) is assigned priority: setPriority(***100 – 9*i***) . The following three methods have been used to vary priority among them:

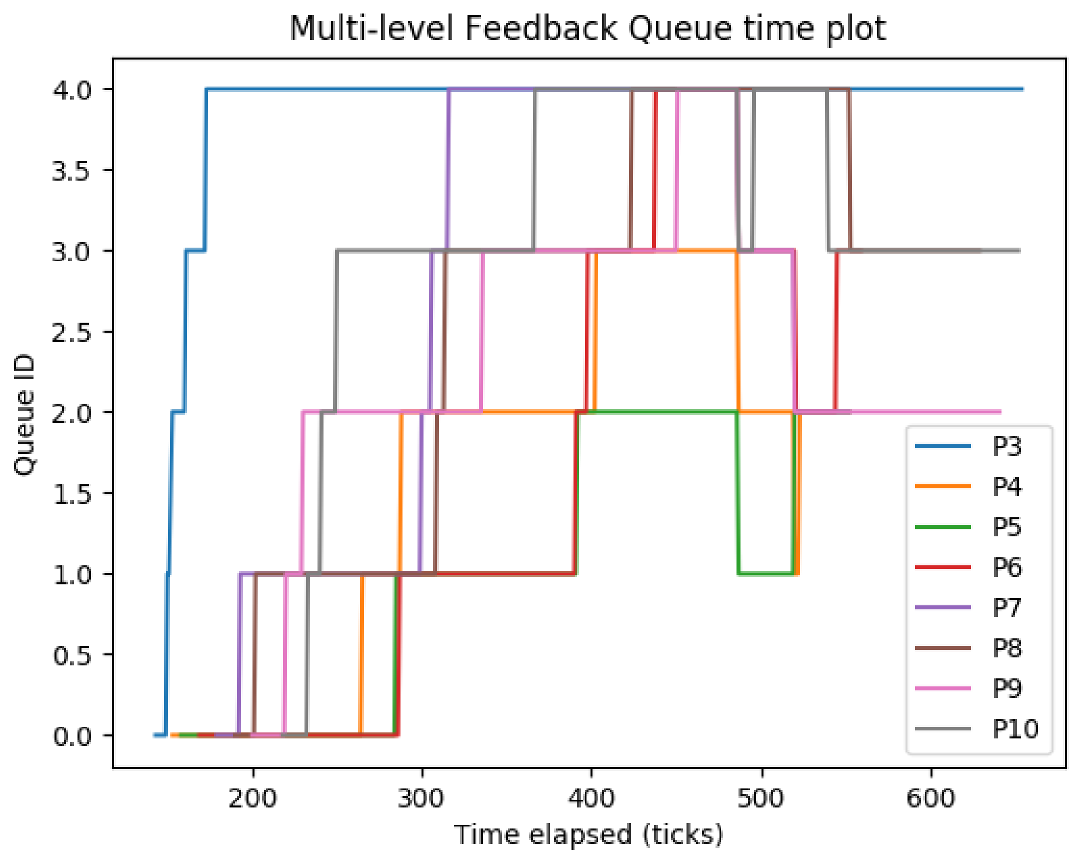
1. The ith process sleeps for ***i*** times and rest of the times (10 - ***i***) uses CPU (relatively high priority to IO intensive processes).
2. The ith process sleeps for ***10 – i*** times and rest of the times (***i***)uses CPU (relatively high priority to CPU intensive processes).
3. The above command (setPriority) is not called so it runs similar to RR which can be verified from the plots show below and above.



Bonus

- Similar to benchmark, bonus.c file has been implemented with less iterations since values are being printed (IO) so huge amount of time is needed to run the program.

- The following plot is shown where 10 child processes have been created by forking. Each child process runs sleep(100) once and then iterates for 1e7 iterations doing both in an alternative fashion. We can observe that as the process 3 begins, it runs out of its time slice so gets promoted to the next queue. Meanwhile, more processes start entering so competition increases and promotion/demotion occurs and aging can also be seen in various processes like P4, P5, P9 towards the end.


Multi-level Feedback Queue time plot
Queue ID
Time elapsed (ticks)
P3
P4
P5
P6
P7
P8
P9
P10