CPU Scheduling with Multiple CPU Bursts

John Wangari

James Naurot

We, _John Wangari_& James Naurot_ (put your names on the line given), have completed this final report and understand that this report is to reflect our own work, unless explicitly and appropriately referenced. Signature: JN & JW

Date of Submission: 12/3/2020

1 Table of Contents

1. INTRODUCTION

Our project aims to create an application to demonstrate multiple cpu scheduling algorithms in conjunction with multiple cpu bursts. We will run simulated scheduling algorithms and use our output to show how each algorithm affects processes and process completion time.
We will examine the effects of the first come first serve, round robin, priority, and multi-level feedback queue algorithms.
We designed the output to show the state of the cpu at each time slice. Our project hopes to bring a clearer understanding of how different algorithms operate and how the different algorithms affect process completion time.

2. IMPLEMENTATION

We decided to go with the algorithms discussed in class as they were known to us and we had tried some of them in labs. The algorithms we used are common but effective. John primarily worked on the Priority Queue and implementation, while James worked on implementing the Multi-Level Queue. We had to collaborate quite a bit since multi-level queue depends heavily upon the priority queue. Additionally, we collaborated on the additional fields necessary for the book-keeping of multiple events for a process.

2.2.1. First Come First Serve

This algorithm takes the process at the head of the ready queue and places it in the running queue until its CPU burst rate has completed. All other processes must wait until the current process has completed its CPU burst rate.

2.2.2. Round-Robin

Round Robin takes the process at the head of the ready queue and places the process in the running queue. Unlike the First Come First Serve algorithm, the process does not necessarily remain in the running queue until process completion. Round Robin utilizes a time quanta whereby, if a process has not completed its CPU burst rate in the allotted time, it will be timed out and placed at the tail of the ready queue. This algorithm assures the each process will be seen by the CPU in less time than for FCFS. For the purposes of this project, we have set the

Round Robin time quanta to be five.

2.2.3. Priority

Each process is randomly assigned a priority. A higher priority, with a higher number being a higher priority, will pre-empt any lower priority process for the running queue. The lower priority process will be returned to the tail of the ready queue.

3. MULTILEVEL SCHEDULING ALGORITHM

The multi-level queue algorithm allots a given time quanta to the process at the head of the ready queue. The process is brought into the running queue and, if the process exceeds the time quanta, the process is placed in a queue of longer time quanta but lower priority. There are three levels of queues in our multi-level queue, so if a process exceeds the time quanta of the second level, it is placed in the lowest level queue having an even greater time quanta. The multi-level queue is priority pre-emptive, thus any higher level process will pre-empt any lower level process. In our implementation, we logically separated the ready queue by priority. The highest priority queue has time quanta 5, the next lowest has time quanta 10, and the lowest hast time quanta 20.

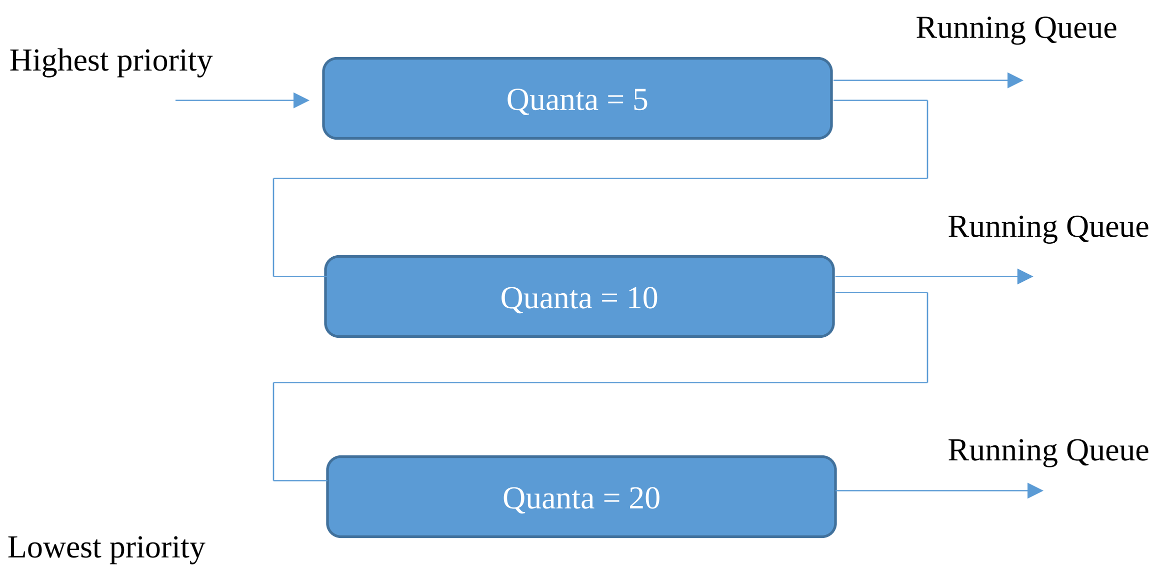

Figure: Multilevel feedback queues

4. CHALLENGES

Implementing the scheduling algorithms, our simulator was very linear. One problem we had was that if a process was in the running queue, it would remain in the running queue even if another process of higher priority had just been brought into the ready queue. We had to add another check before giving the CPU time slice to the current process. This was the only problem we had in the implementation of the program.

5. EXPERIMENTAL SETTINGS

For the purposes of our project, we looked at two processes competing for resources. Much more than two processes became rather confusing. We let the user determine whether to let the computer generate n number of processes or to read processes from a file. Each process must have at least two CPU burst rates with one event in between each CPU burst rate. The event is not to be considered a request for I/O, as we did not implement any device queues.

6. RESULTS

First Come First Serve allowed for the fewest context switches, as evidenced by the output. However, the time till first seen by the CPU was generally not the shortest. FCFS does not equally allocate scarce resources.

Round Robin is fairer in the sense that every process gets a little attention, but not necessarily the fastest in terms of process turn-around time.

Priority doesn’t attempt to be fair, the highest priority gets the resource. In that light, it’s very effective.

Multi-Level Feedback queue attempts to be fair in that every process receives some CPU time; however, CPU bound processes are relegated to lower priority queues so that the scheduler may allocate the CPU to as many short CPU burst rate processes as possible. In this way, throughput of the many processes aren’t held up by the few CPU “hogs”.

7. FUTURE CONSIDERATIONS

Context switching can play a major role in CPU throughput. For instance, FCFS would have very little context switching, but Round Robin would likely have a lot of context switching due to the time quanta. We did not consider context switching and the effects it would have in the various algorithms.

Additionally, we could consider I/O queues and all of the implications they carry, such as deadlock and starvation. We could investigate how to prevent, detect, and resolve deadlock and starvation.

REFERENCES

https://www.tutorialspoint.com/operating_system/os_process_scheduling_algorithms.htm

https://www.pearson.com/us/higher-education/product/Stallings-Operating-Systems-Internals-and-Design-Principles-8th- s

https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/