Differences between scheduling algorithms in a simulated environment

Nathan Hand
nah14@aber.ac.uk

ABSTRACT

In this paper, we discuss the differences between 4 different scheduling algorithm's and the data collected from a simulation of a CPU and a processing queue.

Keywords

scheduling, simulation, algorithms, tasks, jobs, processes

1. INTRODUCTION

The following report outlines the results and information for the CS21120 assignment number 2. The assignment was to use an existing scheduling simulation program with the single implementation of the first come first serve scheduling algorithm. My assignment was to introduce various other scheduling algorithms and then compare results between the different scheduling algorithms. I have introduced the following different scheduling algorithms Round Robin, Priority queue (smallest and largest job first) and Lottery. This report will attempt to answer the following problems/ issues:

- How quickly the first job finished
- The mean time to finish a job
- The overall processing time (CPU time)

As well as take a look at other aspects of the scheduling algorithms.

2. SHORT ALGORITHM DESCTIPTIONS

This section of the report details out a short description of each scheduling algorithm so that you might better understand what to expect from each of them.

First come first served – FCTS simply loads the processes as it receives them into the queue and executes them.

Round Robin – Round Robin gives each process an equal amount of processing time and processes them in a circular order.

Priority queue (smallest) – Also known as Shortest job next essentially sorts the list of processes in shortest job time (cpu requirement) to the head of the queue ahead of larger jobs.

Priority queue (biggest) – This works in the opposite fashion to Shortest job next as the larger processes are given priority over the smaller ones.

Lottery – Lottery assigns each process a random location in the processing queue and then iterates through the queue so shorter and larger jobs could be directly next to each-other each run time.

3. MY HYPOTHESIS

I predict that we will see a clear winner for speed and overall quickest working algorithm and that the algorithm will be "Shortest job first" the reason behind this is idea would be that the shortest job first algorithm seems to make the most sense in terms of being the quickest for turn around in various areas that we're looking at because of the nature of how the algorithm is designed.

3.1 EXPERIEMNT PROCESS

The purpose of this report is to compare and contrast the different scheduling algorithms that I implemented into the simulation program. Several different data files were used for the test and in the case of a random element in the scheduling program 3 tests were run using that scheduler and the same data file. The only scheduling algorithm that used a random element was the Lottery scheduler.

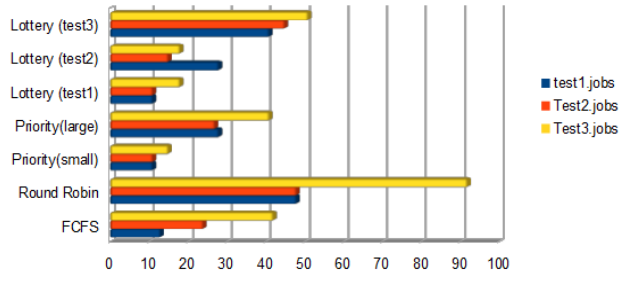
The process was to load each of the 3 data-files into the simulation program running each different algorithm and the results recorded.

3.2 DATA GRAPHS

Due to the nature of my data I want to show the data in an easy to read format. I will be representing the differences between my scheduling algorithms in different graphs and charts for easy reading. The "raw data" straight from the program will be available at the end of this report.

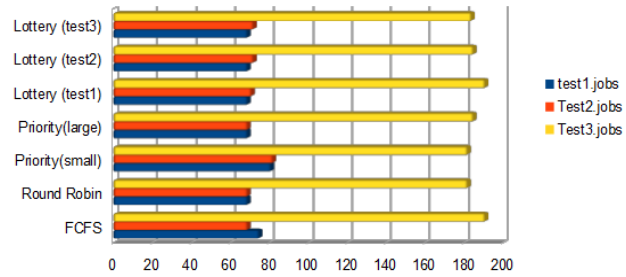
First job time end

Figure 1



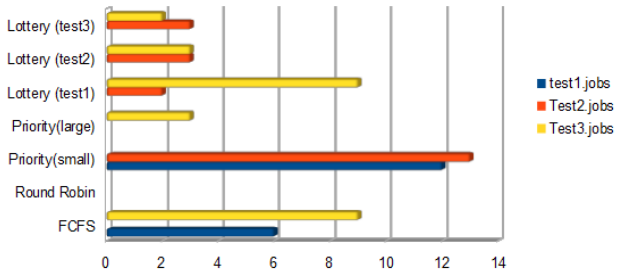
Total CPU time

Figure 2



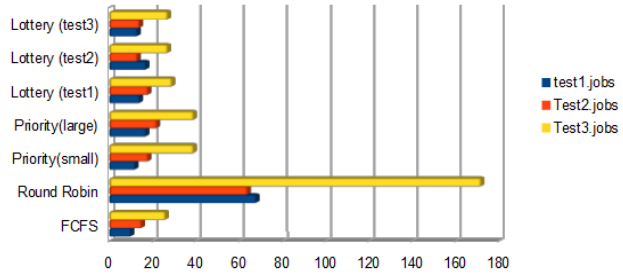
Total Idle time

Figure 3



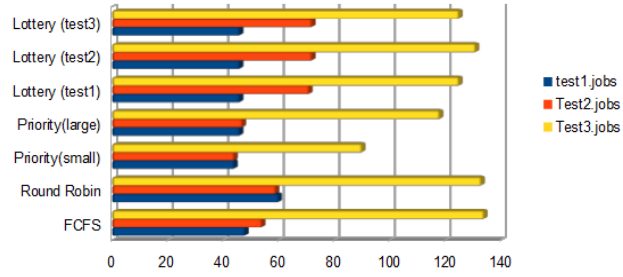
Total Context Switches

Figure 4



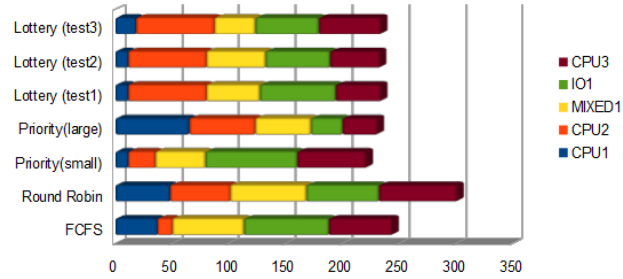
Mean elapsed duration

Figure 5



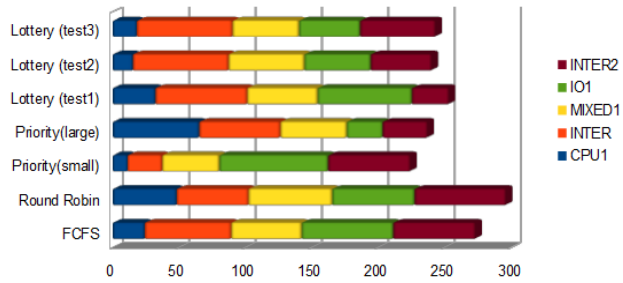
Individual run times test.jobs

Figure 6



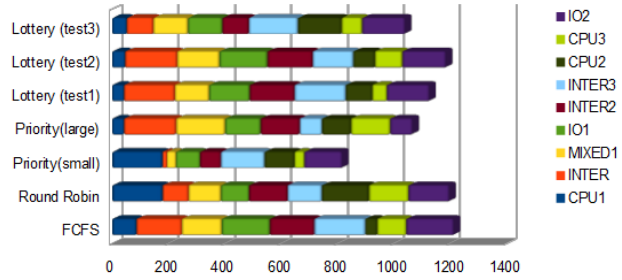
Individual run times test2.jobs

Figure 7



Individual run times test3.jobs

Figure 8



3.3 DATA GRAPHS EXPLAINED

The graphs shown above will be explained In details to the data shown.

3.3.1 Figure 1 – First job finished

Figure 1 shows the amount of CPU time it took for the first job from the queue to finish in its entirety. Certain sheduling algorithms show a clear long turn around time for the first process to finish.

3.3.2 Figure 2 – Total CPU time

Figure 2 shows the total CPU time for the entire simulation for each scheduling algorithm. This data is rather similar across the board.

3.3.3 Figure 3 – Total idle time

Figure 3 shows the total idle time for the entire simulation for each scheduling algorithm. If no bar exists then there was no idle time, this is seen primarily in the round robin scheduler.

3.3.4 Figure 4 – Total context switches

Figure 4 shows the number of "context switches" a context switch is when the scheduling program gives up the current job at the head of the queue and processes another job. This is very prominent in Round Robin due to every single CPU cycle the algorithm switches job.

3.3.5 Figure 5 – Mean elapsed duration

Figure 5 shows the mean for each job for each of the scheduling algorithms, there is a clear distinction between some of the

prioritized scheduling algorithms and the non-prioritized algorithms.

3.3.6 Figure 6 – Individual run times test.jobs

Figure 6 shows the individual run times for each of the jobs seen in the data file “test.jobs” for the sake of the graph all processes are in the same row/ area however for each test the first process to finish does not reflect the order seen here but merely the time it took them to finish.

3.3.7 Figure 7 – Individual run times test2.jobs

Figure 6 shows the individual run times for each of the jobs seen in the data file “test2.jobs” for the sake of the graph all processes are in the same row/ area however for each test the first process to finish does not reflect the order seen here but merely the time it took them to finish.

3.3.8 Figure 8 – Individual run times test3.jobs

Figure 6 shows the individual run times for each of the jobs seen in the data file “test3.jobs” for the sake of the graph all processes are in the same row/ area however for each test the first process to finish does not reflect the order seen here but merely the time it took them to finish.

3.4 HOW QUICKLY THE FIRST JOB FINISHED

The first job to finish is an important process to note as it's the turn around time for the scheduling algorithm. The quicker that the first job is completed for the same job/ process queue shows a great deal about the potential use of a scheduling algorithm for a real program/ process.

This therefore presents excellent findings in terms of each scheduling algorithm. The shortest time to turn around the first job as expected was the priority queue for the smallest job first. This is expected as the smallest tasks are placed at the head of the queue and as such if this scheduling algorithm isn't completing the job ahead of every other scheduling algorithm (with the potential exception of the lottery algorithm) then the smallest job first algorithm would have been implemented incorrectly.

The data collected from my experiment (see Figure 1) clearly shows the best scheduling algorithm for the quickest turn around in first job finished would be the shortest job next algorithm. This algorithm does however create the potential issues of process starvation where the longer process might never get seen too if a large amount of smaller processes are continually added to the processing queue.

These results were definitely expected, for this scheduling algorithm, it is specifically designed with the idea of the first job being completed sooner than any other as it's a prioritized queue for the shortest required CPU time/ job length at the head of the queue.

3.5 THE MEAN TIME TO FINISH THE JOB

The mean time to finish a job is mean of the total processing time between each of the processes. The data collected from my experiment (see figure 5) shows the mean elapsed time between all of the test data files executed. This data therefore indicates the total time it took for each test data file to complete.

There was no expected results from this experiment for a clear winner and that's clearly shown in the data. All of the scheduling algorithms took a similar amount of time to complete the task. However there is a clear difference when we come to test3 which has the largest number of processes of all of the test files. There is a clear indicator that the Shortest job next algorithm is the fastest to turn around that process queue.

This shows that the prioritized queue under shortest job next allows for the quickest mean time.

3.6 OVERALL PROCESSING TIME FOR ALL JOBS

The overall processing time or elapsed real time of all jobs is very similar to the CPU time for the entire simulation. (see Figure 2)

The results shown in Figure 2 clearly show that all of the processes have a similar processing time for each of the test simulations. However I want to pay close attention to the Priority smallest as that has the highest CPU time compared to the rest of the scheduling algorithm for the first 2 simulation tests but again one of the shortest for the larger simulation.

These findings show that for smaller jobs such as test1 and test2 for my simulation more processing power was used to run through the simulation then other scheduling algorithms shown. This data combined with our other findings show a clear indication that although smallest job first is a faster scheduling algorithm it also takes up more CPU for smaller tasks.

While more CPU time isn't a problem for common place computers today this is a real issue when you are dealing with smaller CPU's with a low clock speed such as on a mobile phone, tablet or even small boards such as Arduinos. So while my tests so far have show that shortest time first is a great scheduling algorithm this could be an impossible cost on the CPU that a smaller device wouldn't be able to handle and a less costly algorithm might be required.

These results were expected, the faster the scheduling algorithm for a turn around time of processing is bound to take up more CPU time. The more effective the scheduling algorithm for speed the more power/ CPU time is required.

4. CONCLUSION

I have concluded that different scheduling algorithms have different purposes, so while the better scheduling algorithm we have seen from our various data-sets was clearly the shorted job first algorithm it was the most computationally expensive and might not have been appropriate for different forms of CPU's with limited processing capability.

Round robin proved to be the slower of many tests but also proved to be the least computationally expensive a must in certain environments.

First come first serve proved to be a medium in between the different scheduling algorithms. It had the lowest context switches (see Figure 4) but overall didn't stand out in any other area other than giving the 2nd highest idle time overall.

Largest priority first proved to be an interesting test proving to not actually be a bad scheduling algorithm in certain respects such as

idle time. Idle time for the largest job first scheduling algorithm was actually non-existent for the smaller test files. (see Figure 3)

Lottery proved to be the most unpredictable (due to it's random element) and would generally prove to perform vastly differently on all tests. I don't see any proper implementation for Lottery in a real program where there is no prioritized queue.

5. REFERENCES

I would like to thank Richard Shipman (rcs@aber.ac.uk) for the initial framework that allowed my implementations of different scheduling algorithms.