Process Scheduling Algorithms: Explored using Interactive Visualization

Haranshvir Gujral
(haransh@uw.edu)

Aakash Bang
(aakashpb@uw.edu)

Chinmay Tatwawadi
(chinmayt@uw.edu)

***Abstract* – Task scheduling has been an important aspect of both, humans and automated machines alike. The manner in which humans undertake the completion of tasks is intangible and not governed by any predefined set of rules. On the other hand, tasks performed by machines can be quantified and streamlined to improve their efficiency. In the domain of task scheduling, the particular aspect of 'process scheduling' undertaken by the processor of the modern computer is vital in improving the performance of processors. Process scheduling has been governed by numerous algorithms since its inception, but apart from the computer scientists who design and implement these algorithms they are not easy for a layperson to understand. The lack of interactive visuals is another obstacle towards effectively understanding how these scheduling algorithms work. Our work in this paper intends to aid individuals in visualizing the concept of process scheduling and impart sufficient expertise to schedule a given set of processes with a particular algorithm.**

***Keywords* - Process scheduling algorithms; operating systems; first come first served; shortest job first; round robin.**

I. INTRODUCTION

Understanding process scheduling in computers is an important step to understanding task scheduling in general. A process is just one of the many forms of tasks; others like threads and data flows are similar in nature but governed by different rules. Learning to schedule processes could act as a stepping stone to get a better understanding of how to schedule tasks in a broader sense. Figuring out how process scheduling works could be a tedious task if one is not familiar with certain concepts of computing. Process scheduling involves various technical terms, such as burst time and turnaround time, which although have concise definitions are harder to understand without an example or someone showing how they actually work. There is a large collection of technical papers and manuals available on the internet that touch upon the topic of process scheduling and dive to deeper depths that only computer scientists could make sense out of. The lack of visual representations of the various methods further impairs an individual to learn process scheduling on their own. The modern approach to education involves imparting instruction with the help of visuals to help understand concepts more effectively. Our approach to exploring process scheduling will involve creating visuals and explaining the overall method at the same time. Our work will also serve as a basis for future improvement in this area and it can be applied to other aspects of task scheduling in a similar manner.

In this paper we plan on creating, explaining, and visualizing three methods that are popular in process scheduling. They are called process scheduling algorithms: First Come First Serve (FCFS) scheduling, Shortest Job First (SJF) scheduling, and Round Robin (RR) scheduling are the ones we are going to direct our focus towards. Our work will address the working, merits, and demerits of these algorithms by creating interactive examples and visuals.

II. RELATED WORK

We found many articles that touch upon various topics in the overarching domain of task scheduling. From there, we narrowed our focus to those dealing with process scheduling algorithms and their implementation by computer processors. Most of these articles were authored by computer science professors from various reputed universities across the world. One of the notable ones is a book written by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau called 'Operating Systems: Three easy pieces.' The authors have explained the working of many operating systems concepts such as Virtualization, Concurrency, and Persistence. [1] Their explanation of process scheduling was very helpful for our understanding and this enabled us to conduct our own research in order to use the concepts we had learnt and create effective visual representations in order to impart instruction to others as well. Chapter 7 – Scheduling: Introduction

specifically deals with FCFS, SJF, and RR scheduling policies and also explains the metrics used to evaluate these policies, such as turnaround time and response time.

Workload Assumptions

The authors Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau suggest a few simplifying assumptions for the processes that need to be scheduled. Processes running in a system are collectively known as workload. The terms process and job are interchangeable, and so are scheduler and processor in this context. Determining the workload is a critical part of building policies, and the more you know about workload, the more fine-tuned your policy can be. [1] To start with, the assumptions as suggested by the authors would be slightly unrealistic, but as we progress we will fine-tune them to reach more realistic goals. The following assumptions are suggested by the authors: [1]

1. Each job runs for the same amount of time.
2. All jobs arrive at the same time.
3. Once started, each job runs to completion.
4. All jobs only use the CPU (i.e., they perform no I/O)
5. The run-time of each job is known.

The last one is particularly unrealistic, since a scheduler (here, the processor) would have to be able to look into the future in order to know the run time of each process before it starts executing them, which would be great to have but rather impossible to implement. Later on, SJF will explain this conflict in more detail. Also, as our work progresses, we aim to include processes having variable run-times (also called burst time) since that would be a better representation of real-life examples of processes. Furthermore, jobs once started will run to completion is the basis of FCFS as we will see ahead. Other scheduling policies will not follow this assumption, since the concept of preemption will come into the picture too.

Scheduling metrics

After tackling the assumptions, we need certain criteria to evaluate a scheduling policy and also compare it to the other policies. A metric is something that is used to measure something, and since we are measuring a scheduling policy it can be called as scheduling metrics. Since processes are being measured by the time they take to run, we can apply the same time-based metric to the scheduler as a whole. The turnaround time of a process is the effective time a process was running for in the processor, which can be expanded as the time taken for the process to run to completion minus the time at which it arrived in the system.

$$T_{Turnaround} = T_{completion} - T_{arrival}$$

Since our second assumption stated that all processes arrived in the system at the same time, we can put $T_{arrival} = 0$ in the above equation. Hence the equation would reduce to $T_{Turnaround} = T_{completion}$ for now, but we will see how the equation can be tailored to each scheduling policy separately. The average of the turnaround time for all processes gives us the average turnaround time for the scheduler. Another metric that can be used is waiting time, which is the amount of time a process has to wait in the queue for its turn to be executed. An average of all the waiting times gives us the average waiting time for the scheduler running on a particular scheduling policy, and there will be noticeable differences between these values across the different scheduling policies.

III. METHODS

First Come, First Served (FCFS)

This is the easiest and simplest scheduling policy to implement under our given assumptions. All the processes that arrive in the system are placed in a queue. The processor then executes them in the order of their arrival, which means the one that arrives first is served first, and hence the name of this policy. The processor does not differentiate between the processes in the queue, so this policy appears to be fair. The disadvantage is that this method results in poor average waiting time and consequently poor average turnaround time. [2]

Consider three processes, P1, P2, and P3 that have equal burst times according to our first assumption. Let us say these burst times are 10 milliseconds each. The processor queues them in FCFS manner and begins to execute them as shown below:

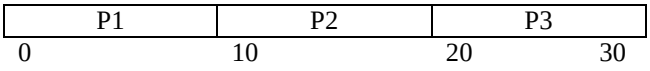

Figure 1: FCFS scheduling with equal burst times

As seen, process P1 finishes at 10, P2 at 20, and P3 at 30. The average turnaround time for these processes can be calculated as (10+20+30)/3 = 20 milliseconds and the average waiting time will be calculated as (0+10+20)/3 = 10 milliseconds for every process. Under our first assumption, it looks like FCFS performs well enough, so we can drop this assumption to explore a scenario where FCFS could perform poorly. Let us consider unequal burst times and change P1 to 100, but keep P2 and P3 as 10 (same as before). The processor queues them in FCFS manner and begins to execute them as shown below:

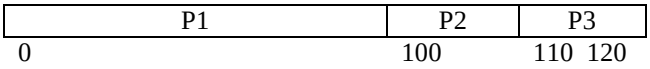

Figure 2: FCFS scheduling with unequal burst time

As seen, process P1 finishes at 100, P2 at 110, and P3 at 120. The average turnaround time for these processes can be calculated as (100+110+120)/3 = 110 milliseconds and the average waiting time will be calculated as (0+100+110)/3 = 70 milliseconds for every process. This time around we ended with very large values for these metrics means poor performance. This problem arises because the relatively short processes P2 and P3 get queued behind longer process P1, and they processor does not factor this in and proceeds to execute them in strict order. This scenario is similar to a the line at the grocery store checkout, where the person in front of you has many more items in their cart and you have only a few, still you would have to wait for a while for them to finish checkout before your turn comes to checkout.

Shortest Job First (SJF)

Now imagine if we could cut the person with the cart full of items in order to finish our checkout of just a few items before them. This would mean significantly less waiting time for us at the cost of a little waiting time for them. This is the basis of Shortest Job First scheduling policy, where the process that has the shortest burst time is executed first, then the next shortest process, and so on until all processes have been executed. Since longer processes are made to wait towards the end, we believe this will result in shorter average waiting time and average turnaround time for the processes. [2] Under our second assumption of all processes arriving at the same time, let the three processes from the previous example be considered once again. The order of arrival is P1, P2, P3 into the system, but the processors is now working with the SJF scheduling policy and therefore selects the shortest process for execution first. Here P2 and P3 both have burst times of 10 milliseconds each so the processor does not have to worry about anything and can execute either one first, or the one which arrived first, that is P2. Either of these decisions will not have an effect on the scheduling metrics that will be calculated ahead. The execution in SJF manner is shown as below:

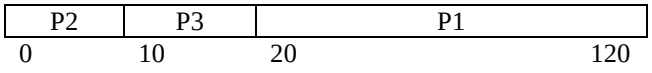

Figure 3: SJF scheduling

The average turnaround time for these processes can be calculated as (10+20+120)/3 = 50 milliseconds and the average waiting time will be calculated as (0+10+20)/3 = 10 milliseconds for every process. This is a significant improvement from the FIFO policy and the average turnaround time has improved by a factor of greater than 2. The assumption that all processes arrive at the same time makes SJF the optimal scheduling policy. [3]

This is true as long as we are considering non-preemptive scheduling, which means that once a process starts executing it is run till completion. Modern process schedulers are based on preemptive algorithms, which are willing to stop the execution of the current process in order to execute another if doing so improves the throughput of processes in the system. This is termed as a context switch, when one process is halted and another process is run. In a scenario where the processes do not arrive at the same time, the longer process could be executed for some time first till the shorter one arrives. The scheduler will then preempt the currently running longer process and perform a context switch that will enable the shorter process to run to completion. This seems like an ideal scenario, but many times it happens that shorter processes that are arriving keep preempting the longer process that arrived before them, and hence the longer process never gets to execute till completion. This scenario is known as starvation, as the longer process is starving since it is not receiving any processor time to execute.

Round Robin (RR)

To solve the problem of starvation, we introduce the Round Robin scheduling algorithm. Suppose in the grocery store scenario, there are multiple people waiting in line to checkout different number of items in their cart. FIFO would make the line proceed in the order of the arrival of customers in line, which doesn't help anyone in case of a processor. SJF would keep serving the customers with few items first and the ones with more items will have to keep waiting till all the newcomers with less items are done with their checkout. In this scenario, Round Robin would mean that the cashier would checkout one item for every customer in the line at a time, and repeat this process. Thus, no customer feels left out, and the line is being shortened one grocery item at a time. Similarly in process scheduling, each process would run for a specific amount of time, known as a time quantum or time slice that is predetermined by the scheduler. [4] Each process runs for exactly one time quantum, and then stops running to allow the next process to run for one time quantum, and so on. This cycle repeats until all processes have been executed. [2]

Consider the three processes P1, P2, P3 where each process wishes to run for 3 milliseconds. The scheduler gives a time quantum of 1 millisecond for each process to run. This is how their execution would look like:

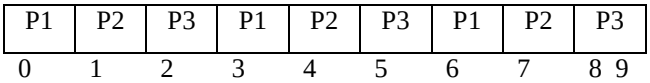

Figure 4: Round Robin scheduling

In the first 3 time quantum periods, all three processes P1, P2, and P3 have been executed partially. This means that

they didn't have to wait for a long time for their turn to start, and this improves the response time of the scheduler. Response time is the difference between the time when the process arrives in the processor and the time when it was first scheduled to execute.

$$T_{Response} = T_{first\ run} - T_{arrival}$$

Each process runs for 1 time quantum and is then preempted by the next process. The scheduler performs a context switch and allows the next process to run, and so on. An SJF scheduler would run each job to completion and the average response time would be (0+3+6)/3 = 3 milliseconds for SJF. A Round Robin scheduler with a time quantum of 1 millisecond would have an average response time of (0+1+2)/3 = 1 millisecond. Thus, round robin performs better in this scheduling metric.

The length of the time quantum is critical for the performance of a Round Robin scheduler. A shorter time quantum improves the response time metric for the scheduler. [5] When the time quantum becomes too small, the number of context switches become too large and the overhead involved with these switches would incur a large cost to the overall performance of the scheduler. Deciding the length of time quantum is therefore a trade-off to the system between improving performance and reducing overhead cost of context switching.

IV. RESULTS

After conducting research on the process scheduling policies and developing a method to make learning these concepts easier, we have visually implemented the scheduling policies. First we demonstrate an example of process scheduling using interactive multimedia player so that the user can grasp the basic concept of how they work. Next, we have also implemented an interactive process scheduler that allows the user to input the burst time values for the processes and see the visual representations of how the processor executes them. This would give the user a better idea about the various cases of process execution and test various cases to see which of the process scheduling policies perform best under a given set of processes.

The example shown in the following figures has three processes P1, P2, and P3 with burst times of 5, 3, 2 milliseconds respectively. The timer shows the current time from the processor's perspective, so as to track the execution of the processes. The user can select the options to play the visual non-stop, or frame by frame using the keys provided in the figure. After playing the visual, the processes are scheduled and executed according to the scheduling policy mentioned. Each process is represented by its name, and is placed inside a cell that is proportional in size to the time taken to execute it (for FCFS and SJF). In Round Robin scheduling, the default time quantum used is 1 millisecond, so each cell will be of equal size. The process highlighted in red is the process that is currently executing. While executing, the corresponding scheduling metric is updated in an equation shown below the cell blocks. For FCFS and SJF, we use average wait time as the scheduling metric, and for Round Robin we use average turnaround time, which is synonymous to the average wait time metric when applied to Round Robin scheduling. After execution is completed, the final value of the corresponding scheduling metric is displayed at the end of the equation.

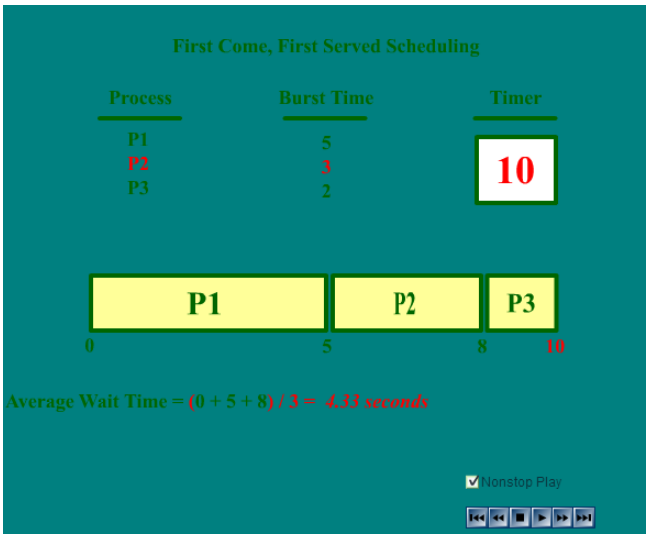

Figure 5: FCFS visualization

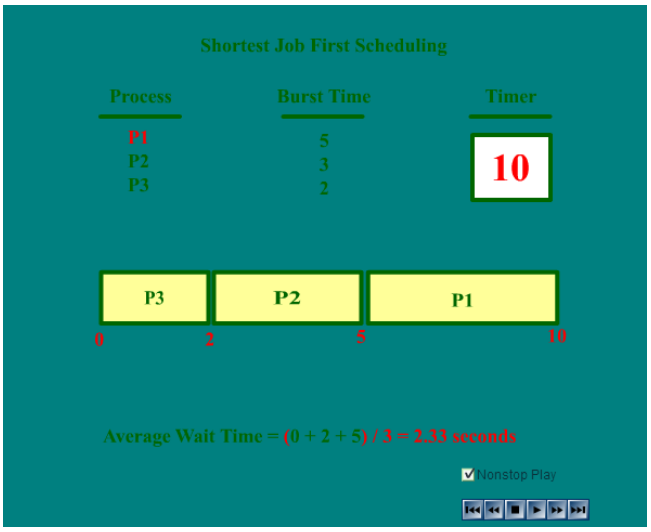

Figure 6: SJF visualization

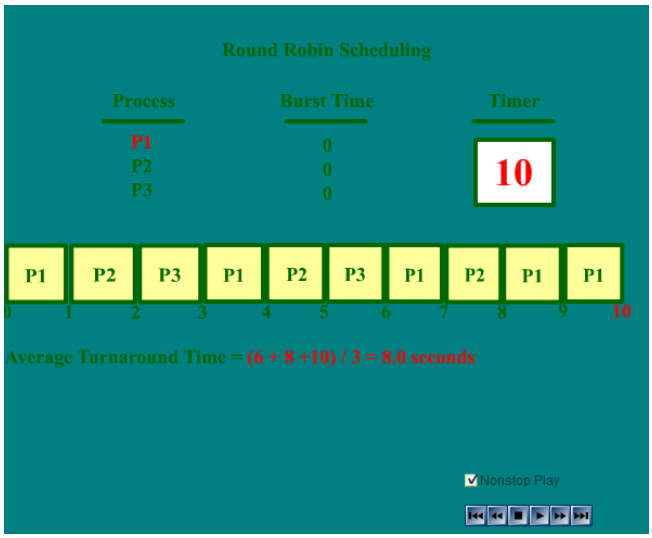

Figure 7: Round Robin visualization

After the user has grasped the basic concept of process scheduling, they can enter the next phase of implementing them using their own inputs. This will enable them to try out different cases with 5 processes, see their execution, and compare the scheduling metrics. First the user is asked to input the burst times for 5 processes so that they may be inserted into the processor queue. Next, the user can select one of the three given scheduling policies in order to execute the given set of processes. After the execution is complete, the user can compare the scheduling metrics for the three policies which is displayed as a bar graph.

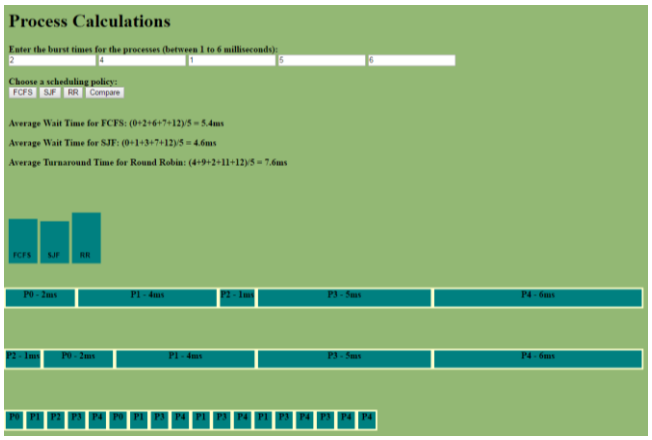

Figure 8: Interactive process calculations

V. DISCUSSION

Our initial objective was to enable the user to learn the various process scheduling algorithms. By providing some background knowledge in the form of this paper, and coupled with the interactive visualizations created, we believe that the user will understand the underlying concepts behind Process Scheduling in computers. By providing the option to execute processes of their own choice, the user has the freedom to explore this topic with ease. Although there is scope for many improvements, this visualization is a much needed first step towards elucidating algorithms such as process scheduling. This can be further expanded to cover more algorithms of the Operating Systems domain that are of prime importance in the modern world of computing.

VI. FUTURE WORK

The topic we are dealing with has a lot of scope for future work, both in terms of size and complexity. Our initial work can be applied to numerous other algorithms within the Operating Systems domain, and include scheduling in multiprocessor environments. Also, many more scheduling metrics can be applied to each algorithm to give the user an even better idea when evaluating these methods. Furthermore, we made workload assumptions in the start that eased our implementation of the algorithms. Dropping each assumption adds a certain level of complexity to the implementation, and such a task can only be accomplished over a larger period of time.

For the future, we plan to include certain new algorithms such as Priority Scheduling, which is increasingly popular in the domain. In terms of our implementation, we would like to add features such as: allowing the user to select the number of processes that have to be executed; selecting the scheduling metric that needs to be compared; enabling the user to set the length of the time quantum in Round Robin; and so on.

REFERENCES

1. Remzi H. Arpaci-Dusseau; Andrea C. Arpaci-Dusseau (2015). "Chapter 7: Scheduling: Introduction, Section 7.6: A New Metric: Response Time". *Operating Systems: Three Easy Pieces*. Retrieved March 7, 2016.
2. Stallings, W. (2001). *Operating systems: Internals and design principles* (4th ed., Stallings, William. William Stallings books on computer and data communications technology). Upper Saddle River, N.J.: Prentice Hall.
3. (n.d.). Retrieved March 14, 2016, from https://en.wikipedia.org/wiki/Scheduling_(computing)
4. Process Scheduling. (n.d.). Retrieved March 15, 2016, from https://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html
5. Scheduling Algorithms. (n.d.). Retrieved March 15, 2016, from http://www.gitam.edu/eresource/comp/gvr(os)/5.3.htm