Fairness in CPU Scheduling: A Probabilistic Algorithm

Anagha H C
211IT008
Department of Information Technology, National Institute of Technology Karnataka, Surathkal

Sachin Prasanna
211IT058
Department of Information Technology, National Institute of Technology Karnataka, Surathkal

Abhayjit Singh Gulati
211IT085
Department of Information Technology, National Institute of Technology Karnataka, Surathkal

***Abstract*—This paper presents a novel CPU scheduling algorithm for a uniprocessor system that utilizes a probabilistic function to provide fair resource allocation to the CPU. In this paper, the proposed algorithm is described in detail, including its implementation and comparison with existing algorithms based on performance metrics. The various metrics used for comparison are average turnaround time, average waiting time, CPU utilization, fairness, and throughput. The results of the performance evaluation demonstrate the effectiveness and drawbacks of the proposed algorithm in achieving fair resource allocation in a uniprocessor system. This paper contributes to the field of operating systems by providing a new CPU scheduling algorithm that addresses the need for fair allocation of resources, with taking into regard priority of processes as well.**

Keywords—Probability, Priority, Fairness, CPU Scheduling, Resource Allocation, Probabilistic Fair CPU Scheduling, PBF, Gini Coefficient

I. INTRODUCTION

In modern operating systems, the CPU scheduling algorithm plays a critical role in allocating the available system resources to processes efficiently. The CPU is one of the most valuable resources in a computer system, and efficient utilization of the CPU is essential for optimal system performance. In a uniprocessor system, the CPU scheduling algorithm is responsible for selecting the next process to run on the CPU. This decision has a significant impact on system performance, as it directly affects the responsiveness, throughput, and fairness of the system. Several CPU scheduling algorithms have been proposed in the literature, each with its strengths and weaknesses. One of the earliest algorithms is the First-Come, First-Served (FCFS) algorithm, which schedules processes in the order they arrive in the ready queue. While FCFS is easy to implement and ensures fairness, it suffers from poor performance due to long waiting times and inefficient use of the CPU. Other algorithms, such as Shortest Job First (SJF) and Round Robin (RR), aim to improve performance by prioritizing short jobs or allocating CPU time in a time-sliced manner, respectively. These algorithms have shown significant improvements in performance compared to FCFS, but they still have limitations in terms of fairness and responsiveness. Therefore, there is a need for new CPU scheduling algorithms that can provide fair allocation of resources while maintaining optimal system performance and keeping in mind that certain processes are more important than others and need priority. In this paper, we propose a novel CPU scheduling algorithm that utilizes a probabilistic function to achieve fair resource allocation in a uniprocessor system. We discuss the idea and implementation of our algorithm and compare its performance with existing algorithms. We call this new algorithm the **Probabilistic Fair CPU Scheduling Algorithm** or **PBF** in short.

II. CPU SCHEDULING CRITERIA

Different CPU scheduling algorithms possess distinct properties, and selecting a specific algorithm may give preference to one type of process over another. In order to determine which algorithm to use in each situation, it is essential to consider the characteristics of each algorithm. Numerous criteria have been proposed for comparing CPU scheduling algorithms, and the choice of which criteria to use can have a significant impact on determining the best algorithm. These criteria include:

- **Utilization/Efficiency:** This criterion aims to ensure that the CPU is utilized 100% of the time with useful work.
- **Throughput:** This criterion focuses on maximizing the number of jobs processed per hour.
- **Turnaround time:** The time from submission to completion should be minimized to reduce the waiting time for batch users for output.
- **Waiting time:** This criterion aims to minimize the sum of times spent in the ready queue.
- **Response Time:** The time taken from submission to the first response should be minimized to ensure minimum response time for interactive users.
- **Fairness:** This criterion aims to ensure that each process gets a fair share of the CPU.

III. IDEA

The idea behind the algorithm was to give priorities to certain processes, while also ensuring that they do not occupy the CPU for a longer duration of time, which starves other processes from getting the CPU. We intertwine the concepts of time quantum, probability, and priority to achieve our goal. The fairness of allocation of resources is improved by introducing a probability function along with the concept of priority. The involved formula is:

$$Probability = \frac{Priority + k}{TotalPriority + (n * k)}$$

Where:

- **Probability** is the probability assigned to a particular process which is present in the Ready Queue.
- **Priority** is the priority of the process being considered.
- **k** is an adjustable parameter. An increase in the value of k normalises the priority values, and disregards priority if required. A decrease in k makes the function a basic probability function, and makes the difference in priorities more evident.
- **TotalPriority** is the sum of priorities of all processes in the ready queue.
- **n** is the number of processes in the ready queue.

Having a probability rather than plain priority ensures that every process has a chance of being allocated to the CPU. The algorithm also involves a time quantum. The introduction of time quantum ensures that no process uses the CPU for an increased share of time. Once the time quantum expires, the process is reallocated to the ready queue. If the process has exhausted its burst time before the end of its time quantum, a algorithm of picking the next process is run again.

IV. ALGORITHM

- Initialize the process arrival time, burst time, and priority in arrays.
- Initialize a map for each process, representing whether it is currently in the ready queue. This map plays the role of the ready queue.
- Calculate the initial minimum arrival time. The processes (or process) which have (has) minimum arrival time are added to the ready queue and process of selecting a process is done.

Loop until all processes have been terminated:

1. Run the process picked until the time quantum, or till its remaining burst time (if it is lesser than the time quantum) and advance the time accordingly. If the process is completed, i.e., burst time remaining is 0, then set its completion time to the current time, and set its value in the map to false. This indicates that is has been removed from the ready queue.

2. Check if there are any processes with the arrival time lesser than the current time and add them to the ready queue by setting their map value to true.

3. Select the next process to run using the selectProcess function, which calculates the probability of each process based on its priority and the total priority of all processes currently in the queue, and then selects a process at random.
4. If the ready queue is empty, but not all processes have terminated then find the process (processes) which has (have) minimum arrival time and with a burst time greater than 0 and run the process of selection again.

5. If the ready queue is empty and all processes have completed, then break out of the loop and calculate the turnaround times and waiting times according the standard formulae:

$$Turnaround\ time = Completion\ time - Arrival\ Time$$

$$Waiting\ Time = Turnaround\ time - Burst\ Time$$

The **selectProcess** function when called, adds the priority of all processes currently in the ready queue and the number of processes in the queue.
Then, a probability is assigned to each process according to the formula mentioned above.
Finally, a process is picked in accordance with its probability and given to the CPU for execution.

V. IMPLEMENTATION

The algorithm was first implemented in C++ and tested for correctness. After verification, a user interface was built where the user could input the arrival times, burst times, priorities, time quantum and the adjustable parameter k. On submission, the output table and the Gantt chart would be displayed.
The user interface was built using HTML, CSS, and JavaScript. The program consists of 3 main parts:

1. **HTML:** For user input and the skeleton of the interface.
2. **CSS:** For styling the interface and making it look more presentable and easier to use.
3. **JavaScript:** Implementation of the algorithm.

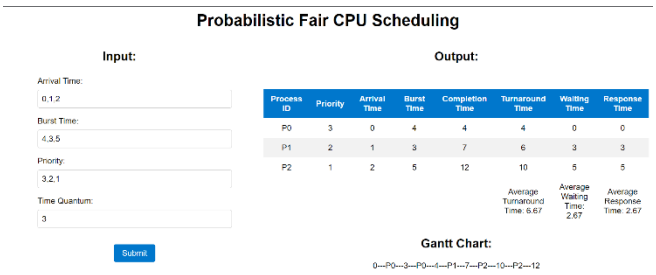

Figure 1: User Interface

VI. ANALYSIS

The algorithm was analyzed four metrics – Response Time, Waiting Time, Turnaround Time and Fairness.

Metrics such as throughput and CPU utilization need kernel level coding and actual implementation to the kernel, which can be a study for the future.

All the metrics were calculated based on 6 different situations. The first 3 situations are applicable for all algorithms, whereas the last 3 situations are only applicable for priority-based algorithms. The situations are:

1. The process which needs longer time comes first.
2. The process which needs shorter time comes first.
3. The process which needs medium time comes first.
4. Higher priority processes come first.
5. Lower priority processes come first.
6. Medium priority processes come first.

A. RESPONSE TIME

Algorithm	Shortest Processes First	Medium Processes First	Longest Processes First
FCFS	15.66	34.66	42.33
SJF	15.66	17.33	23.33
SRJF	15.66	17.33	8.77
RR	7.11	10.66	11.11
PNP	24.88	22.22	24
PP	24.66	19.22	0
PBF	**10.23**	**13.67**	**15.78**

Only second to the Round Robin Algorithm, the Probabilistic Fair Algorithm offers good response times compared to other algorithms. Although, for the priority-based algorithms, the data can be a bit misleading, so a better comparison is presented in the table below:

Algorithm	Lower Priority First	Medium Priority First	Higher Priority First
PNP	11.83	11.83	21.33
PP	11.16	8.83	0
PBF	**9.82**	**11.18**	**11.56**

The Probabilistic Fair algorithm fares decently with respect to the other two priority-based algorithms. Although, there is scope of improvement with different variations in time quantum.

B. WAITING TIME

Algorithm	Shortest Processes First	Medium Processes First	Longest Processes First
FCFS	17.89	34.66	42.33
SJF	16.95	17.33	23.33
SRJF	15.67	15.43	19.11
RR	27.67	34.77	39.33
PNP	17.78	35.55	42.33
PP	21.44	38.11	41.89
PBF	**23.45**	**31.81**	**36.54**

FCFS – First Come First Serve
SJF – Shortest Job First
SRJF – Shortest Remaining Job First
RR – Round Robin
PNP – Priority non-Preemptive
PP – Priority Preemptive
PBF – Probabilistic Fair

In all 3 scenarios, the Probabilistic Fair Algorithm had a lesser average waiting time than the Round Robin algorithm. It had better times than the priority-based algorithms as well when the medium and longest processes come first.

Algorithm	Higher Priority First	Lower Priority First	Medium Priority First
PNP	16.66	14	16.16
PP	16.66	19.16	17.83
PBF	**18.66**	**18.39**	**18.75**

It was observed that the algorithm suffered a drawback here, as its average waiting time was worser than the other two priority-based algorithms.

C. TURNAROUND TIME

Algorithm	Shortest Processes First	Medium Processes First	Longest Processes First
FCFS	24	43	50.66
SJF	22	25.66	31.66
SRJF	22	25.66	27.44
RR	36	43.11	48.33
PP	26.11	43.89	50.66
PNP	29.77	46.44	50.66
PBF	**30.78**	**40.15**	**44.79**

A similar trend is obtained again in accordance to the average waiting time.

Algorithm	Higher Priority First	Lower Priority First	Medium Priority First
PNP	23.16	20.5	22.66
PP	24.16	25.66	24.33
PBF	**25.67**	**21.34**	**23.45**

Notably, the algorithm fares well when lower and medium priority processes come first, but is slightly off when higher priority processes come first.

D. FAIRNESS

To measure fairness quantitatively, a measure called the **Gini coefficient** was used. In terms of fairness of allocation of resources, a lesser Gini coefficient translates to an algorithm which is better in terms of fairness and means that the algorithm allows each process a fair share of CPU time without starving any process too much. The graph-based results are displayed are shown below. The Probabilistic Fair Algorithm performed extremely well in terms of fairness compared to the other algorithms.

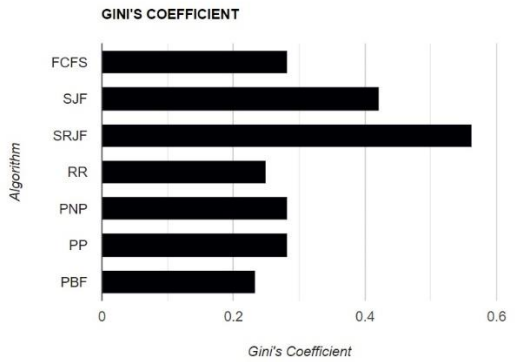

Figure 2: Analyzing fairness using Gini's Coefficient

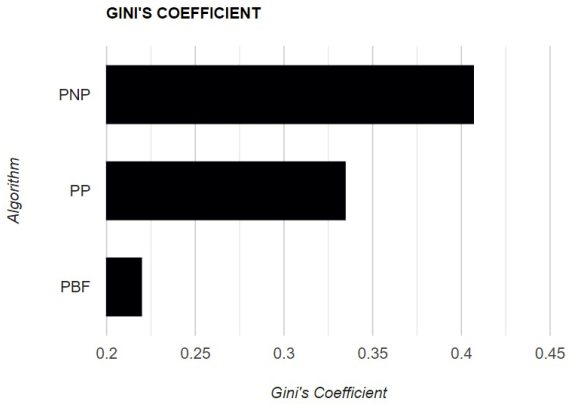

Figure 3: Analyzing fairness using Gini's Coefficient

VII. CONCLUSION

In conclusion, the Probabilistic Fair CPU Scheduling Algorithm offers a novel approach to managing process execution in a fair and efficient manner. Through the utilization of probabilistic calculations and priority-based selection, it aims to strike a balance between fairness and responsiveness in a multitasking environment. Some of the advantages and disadvantages are described below:

A. ADVANTAGES

Fairness: PBF considers the priority of processes when selecting the next process to execute, ensuring that higher-priority tasks receive proportionately more CPU time. Since time quantum is also included, it makes sure that CPU is not monopolized by a single process. Experimental results also backed the fairness of the algorithm.

Responsiveness: By incorporating probabilistic calculations, PBF introduces an element of randomness in process selection. This enables the algorithm to dynamically adapt to workload variations and prioritize processes with higher chances of completion, resulting in improved responsiveness and reduced average response times. The results proved that the algorithm was very responsive to new processes and had good average response times.

Preemptive Time Quantum and adjustable parameter k: PBF incorporates a preemptive time quantum, allowing processes to execute for a limited duration before being reevaluated for fairness and potential preemption and also an adjustable parameter k, which can be altered as per the user's requirements. This makes the algorithm very versatile and makes the algorithm very usable in multiple situations, combining advantages of other already existing algorithms.

Unshackled from Traditional Constraints: In terms of assumptions, PBF aims to minimize reliance on certain assumptions that are commonly found in traditional CPU scheduling algorithms. Algorithms like SJF and SRJF assumes that the burst times of all processes are known in advance, but this is not possible in a realistic scenario and hence an approximate is taken. HRRN assumes that the arrival times of the processes are known and that the system can accurately measure the response time of each process. But, PBF offers a more flexible and robust scheduling solution that can accommodate a wider range of scenarios without heavily relying on specific assumptions.

B. DISADVANTAGES

Overhead: The probabilistic calculations involved in process selection add computational overhead to the scheduling algorithm. While the impact may be minimal for smaller process sets, larger-scale systems with numerous processes could experience increased overhead, potentially affecting overall system performance.

Sensitivity to Parameter Selection: The performance of PBF is influenced by the choice of parameters, time quantum and the adjustable parameter. Selecting optimal parameter values can be challenging and may require fine-tuning to achieve desired results. Poor parameter selection could lead to unfairness, increased response times, or decreased overall system performance. Also,

Randomness: The introduction of probabilistic calculations increases the randomness of the scheduling algorithm. One cannot predict which process will run at a particular time, which may cause some difficulty in scheduling processes in advance in accordance to the time they should run.

In summary, the Probabilistic Fair Scheduling algorithm presents a promising approach to CPU scheduling, offering fairness and responsiveness in a multitasking environment. While it introduces computational overhead and complexity, proper parameter selection and careful implementation can help mitigate these challenges. Further research and experimentation is needed to evaluate the algorithm's performance under various workload scenarios and system configurations, ultimately determining its suitability for real-world applications

VIII. FUTURE SCOPE

The Probabilistic Fair Scheduling algorithm holds potential for future developments and enhancements. Some possible future scopes for the algorithm include:

Performance Optimization: Further research can focus on optimizing the performance of the algorithm by fine-tuning the parameters and refining the probabilistic calculations. Finding the optimum time quantum and adjustable value of k based on lakhs of inputs can be the end goal of the algorithm, which eliminates a lot of redundant possibilities. It would also lead to improved fairness, reduced response times, and enhanced overall system efficiency.

Dynamic Parameter Adaptation: Another interesting take on exploring the algorithm further would be based on dynamic parameter adaptation and time quantum rather than finding the optimal parameter and time quantum. Exploring adaptive techniques to dynamically adjust the algorithm's parameters based on the workload and system conditions could enhance its adaptability. It may also help maintain fairness and responsiveness even in dynamic and unpredictable environments.

Real-time Scheduling: Investigating the feasibility of implementing the algorithm for real-time systems can be an interesting avenue for future exploration. Real-time applications have strict timing constraints, and adapting the Probabilistic Fair Scheduling algorithm to meet those requirements would be valuable.

Resource Constraints Consideration: Extending the algorithm to consider resource constraints, such as limited CPU resources or varying resource availability, can be a future scope. Incorporating resource-awareness into the scheduling decisions can result in better resource utilization and system performance.

Machine Learning Integration: Exploring the integration of machine learning techniques with the Probabilistic Fair Scheduling algorithm can be an intriguing area of research. Machine learning models can learn from historical data to predict process behaviour, arrival patterns, optimal parameter settings and optimal

time quantum further enhancing the algorithm's effectiveness.

IX. INDIVIDUAL CONTRIBUTIONS

A) Anagha HC : Implementation, Analysis, Report, Presentation (??)
B) Sachin Prasanna: Implementation, Analysis, Report, Presentation (??)
C) Abhayjit Singh Gulati: Implementation, Report, Presentation, Analysis (??)

X. ACKNOWLEDGEMENTS

This project titled “Fairness in CPU Scheduling: A Probabilistic Algorithm” was an opportunity for us to explore the field of CPU Scheduling, advantages and disadvantages of several algorithms and develop an algorithm which will address the disadvantages. We would like to extend our heartfelt thanks to Dr. Geetha V. for providing us with invaluable guidance and support throughout the course of this project. Her constant encouragement and constructive feedback helped us stay focused and motivated. We would also like to express our sincere gratitude to our midsem project evalator (??) for his valuable inputs which made us align our thinking in the right way .

REFERENCES

[1] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne: Operating System Concepts, 10th Edition. Wiley 2018