Real Time Scheduling Simulation

Akash Shinde
High Integrity Systems
Frankfurt University of Applied Sciences
akash.shinde@stud.fra-uas.de

Abstract

Understanding the working of scheduling algorithms can help us to enhance the analysis of scheduling processes, resource utilization and performance in real-time applications. Depending on the situation and application, various algorithms function differently and have their own set of qualities that are advantageous. We can see these features, how scheduling algorithms function, and their behavior using a simulator. By automating the processes of scheduling the tasks that are provided as input and easily displaying the output, less work is required to create the schedule, allowing researchers to concentrate on analyzing the behavior of the scheduling algorithms. This research project focuses on creating a web application that is independent of platforms and machines, with the aim of graphically representing various scheduling algorithms to make comparisons and draw conclusions based on the results.

Keywords: Scheduling Algorithms, Real-Time Application, Simulation

1 INTRODUCTION

In several fields, real-time technologies are widely used and in demand. This is because the system delivers information or results within a set amount of time. The systems can be further categorized based on this time constraint as Hard Real-Time Systems, which cannot miss their deadline and would have dangerous impacts if they did and Soft real-time systems with an acceptable probability of rarely missing the deadline. Missing these deadlines wouldn't have disastrous consequences.

The capacity of these systems to meet deadlines is heavily reliant on the scheduler, which optimally schedules jobs and resources for execution. The scheduler is responsible for task completion on time, resource management for job completion, and task feasibility determination. For the operation of this scheduler, many scheduling algorithms satisfy the purpose depending on the necessity.

Scheduling algorithms can be preemptive or non-preemptive. In preemptive algorithms, the task is assigned a specified time-slot and the execution of that task is carried within that time-frame irrespective of the task can be completed or not. If the execution does not finish within the time limit, it is interrupted for the following job. In non-preemptive scheduling, the task must wait for execution until the current task is completed.

The scheduling process can be visualized using various algorithms in the form of a graph, taking into account each unit time and the task that is being executed at that time. Considering this approach into account, this work proposes a technique for simulating various scheduling algorithms and displaying them in a graphical format for a better comprehension of the process.

2 STATE OF THE ART

There are a lot of real-time scheduling simulators available. The majority of these applications display the scheduling process or the feasibility of the tasks for a specific algorithm and are typically executed as a program, with the resulting output of the scheduling process displayed in the console, making it difficult to understand how the tasks were executed during the process and which algorithm executes better. A graphical representation of an algorithm's execution trail provides us with more information to analyze the algorithm's behavior. Even with this graphical depiction, there are only a few simulators that are easily accessible and are created using MATLAB or other platforms, the bulk of which are desktop software, making them machine- dependant.

The proposed simulator focuses on the scheduling and execution of user-provided tasks, applying different scheduling algorithms and presenting intuitive execution results. The web application provides the flexibility of being machine-independent, as well as many possibilities for developing the system and customizing it to meet the needs.

3 REQUIREMENTS AND ANALYSIS

The following functional and non-functional requirements are drawn with the simulator's usability and purpose in mind. This gives an outline of what the simulator should achieve once it is implemented.

3.1 FUNCTIONAL REQUIREMENTS

- User must be able to create, edit and delete tasks and also its.
- Input to the simulator can also be provided via an Excel spreadsheet.
- If an algorithm cannot schedule a task, the simulator must notify the user.

- The simulator must run all scheduling algorithms for the same input at the same time.
- In order to reproduce the real-time scenario, the algorithms' implementation must account for arrival time.
- The execution time of each method must be determined by the LCM calculated using the task duration.

3.2 NON-FUNCTIONAL REQUIREMENTS

- The simulator must present the graphical visualization of all scheduling algorithms.
- Simple User Interface for modifying the task parameters.
- The simulation view must be simple to understand.

4 ARCHITECTURE

- The user interface is displayed on the top level, and it interacts with the React Router, which manages the different views or pages of the application.
- The React components use state and props to manage the data and pass it around. The scheduling algorithms, which includes data processing and application logic, is located in the next layer.
- Lastly an output array generated from these algorithms are passed to components and get rendered in real-time.

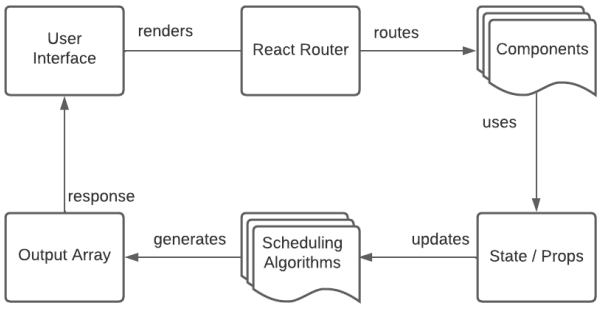

Figure 1: Architecture

In the simulator, the components are files that consists the scheduling algorithm implementation which generates corresponding output array.

5 IMPLEMENTATION

As Shown in the Figure 2, all files under simulationView folder are scheduling algorithm implementations. Single file under common folder is a common hook created to serve as a button. App.js is the main file of this application which handles the routing. The index.html file inside public and index.css inside src are responsible to render the whole application in real-time. Finally, dynamic behaviour of the application is handled by index.js file under src folder.

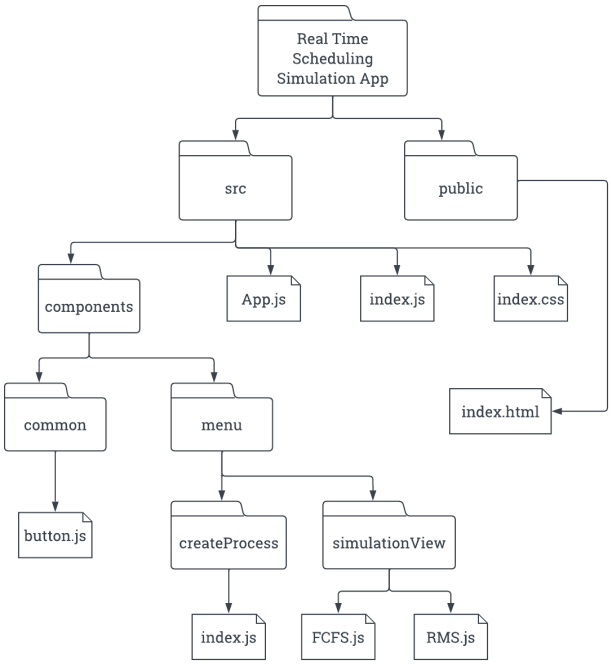

Figure 2: Architecture

Each algorithm is implemented in the application based on the arrival, period, execution time, and LCM(range). The application displays the execution of the process throughout the whole range, thus if a process has an arrival time of 0, execution time of 5, and period of 10, it will be displayed. This process is expected to occur every 10 units until the LCM is reached, according to the application.

5.1 First Come First Serve (FCFS)

The first come, first served algorithm is one of the simplest scheduling algorithms since its implementation is straightforward, as the only parameter needed for scheduling is the process's arrival time. The algorithm prioritizes the processes based on their arrival times, with the one with the shortest arrival time being conducted first. The algorithm is non-preemptive, which means that no other processes can interrupt it; the next process can begin only after the preceding one has finished. Similar to how the queue data structure is implemented. The First Come First Serve algorithm never fails to schedule; it always succeeds in scheduling.

`Algorithm`

1. The arrival time, execution time, and period parameters are considered as inputs to the algorithm.
2. The LCM is calculated based on the process period, or the LCM entered on the UI is used.
3. For execution time, the process with the shortest arrival time is executed first.
4. When this procedure is finished, the next process with the shortest arrival time is chosen.

5. Repeat steps 3 and 4 until the algorithm completes the execution to the LCM.

6. The wait time for a process k is computed using,

```
waitingTime[k] = executionTime[k-1] +
          waitingTime[k-1]
```

   waitingTime[0] = 0, as the first process never waits, its executed as soon as it arrives.

7. The Average Waiting Time can be estimated using the formula,

```
averageWaitingTime =
totalWaitingTimeForAllProcesses /
        numberOfProcesses
```

8. The turnaround time for a process k is computed as follows,

```
turnAroundTime[k] = executionTime[k] +
          waitingTime[k]
```

9. Overall Turnaround Time is computed as follows,

```
averageTurnAroundTime =
totalTurnAroundTimeForAllProcesses /
        numberOfProcess
```

Advantages:

- Simplest Scheduling Algorithm.
- Implementation not complicated.
- First come First Served and allows the task to complete its execution, there is no need for any preemptions.

Disadvantages:

- As the algorithm is Non Preemptive, the aver- age waiting time is very high.
- Short processes that are at the back of the queue have to wait for the long process at the front to finish.
- No Concurrent process execution.
- FCFS is not very efficient.

5.2 Rate Monotonic Scheduling (RMS)

RMS (Rate Monotonic Scheduling) is a preemptive algorithm. It is a priority-based algorithm in which a fixed priority is assigned based on the period. The task that requires the least amount of time will be given the greatest priority. The process with the highest priority will take precedence over any other process with a lower priority. In RMS, deadlines are considered to be equal to periods. There is no resource sharing between the processes. RMS is indeed a scheduling algorithm that is widely used.

Algorithm

1. The RMS method takes as input the execution time, period, and arrival time.
2. The procedure is arranged in ascending order by period. This will assign the work with the shortest time frame and the highest priority.
3. Utilization is calculated to ensure job scheduleability. The utilization for a task set with n tasks is computed as follows,

$$U = \sum_{i=1}^{n} \frac{executionTime[i]}{period[i]}$$

4. If the utilisation $U \leq 1$, which is required for the method to fulfill the CPU utilization, then tasks set can be scheduled.
5. $U \leq n(2^{\frac{1}{2}} - 1)$ is the basic requirement for RMS schedulability. If this criteria is not met, the processes in the current simulator are not schedulable.
6. A list of all the tasks that need to be done based on priority is stored, and this list includes the task's execution time, with the task number serving as the index and the value representing the execution time, and it is updated as the timeline proceeds from 0 to LCM.
7. The tasks are executed once the execution list has been updated and is not empty. The execution time of that process is decremented with each execution until it reaches zero or is preempted by another task if it has a higher priority than the current task being run.
8. This process continues until the time meets the determined LCM.

Advantages:

- It is a simple and easy-to-implement approach that outperforms existing static priority algorithms.
- It contains computed time intervals, as opposed to other methods that ignore job scheduling needs.

Disadvantages:

- It is not optimal in the situation of varying deadlines and time spans.
- Difficult to handle intermittent and periodic tasks.

6 Results & Discussion

Figure 3 refers to a page that displays two algorithm options to select from for scheduling processes in our application. The two algorithms are FCFS (First Come First Serve) and RMS (Rate Monotonic Scheduling), which are commonly used in real-time systems.

Figure 4 refers to a page that allows users to create and input the details of a process they want to schedule using one of the

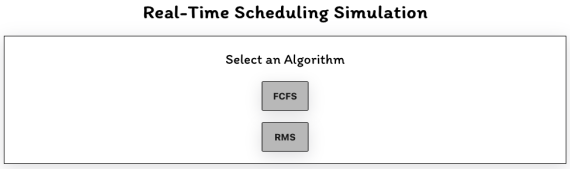

Figure 3: Home Page

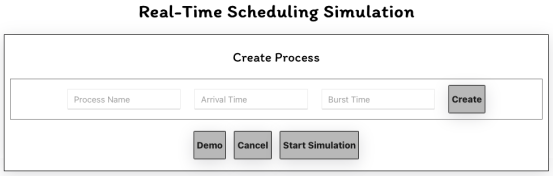

Figure 4: Create Process Page

two algorithms. Users will typically input information such as the name, arrival time, and execution time of the process.

After creating a processes using create process page, the user will be redirected to page in Figure 5. This page typically displays the details of the process created, such as the name, arrival time, execution time, and any other relevant information.

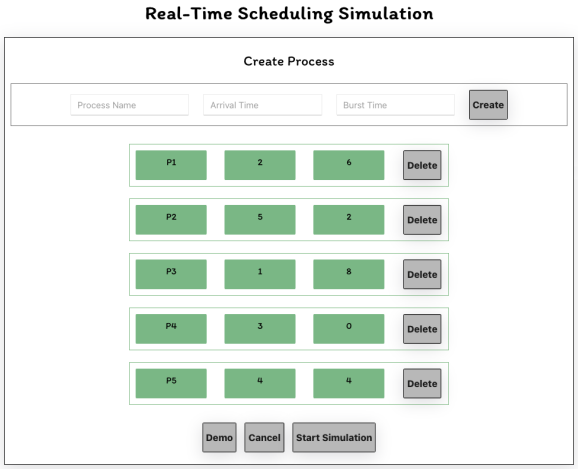

Figure 5: FCFS Processes

Figure 6 refers to the output of the FCFS algorithm after simulating the scheduling of processes. The page will typically show the order in which the processes were executed, their waiting times, algorithm name, average waiting time, average turnaround time, and total execution time.

Similar to the FCFS process created page, Figure 7 displays the details of a process created for RMS algorithm.

Figure 8 displays the output of the RMS algorithm after

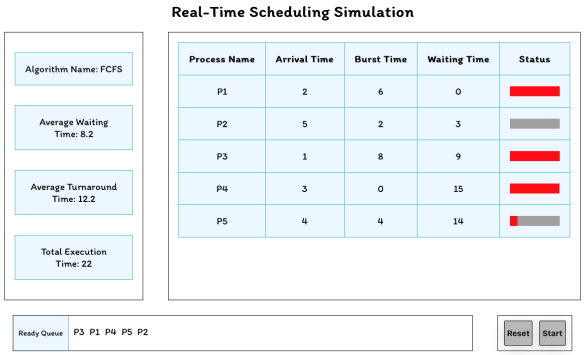

Figure 6: FCFS Simulation

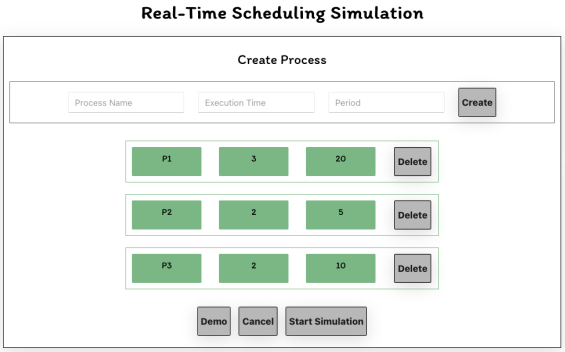

Figure 7: RMS Processes

simulating the scheduling of processes. It will typically show the order in which the processes were executed. The RMS algorithm aims to meet all process deadlines, and if it is unable to do so, an error will be displayed.

Finally Figure 9 is displayed if the RMS algorithm is unable to meet all process deadlines. This error occurs when the system is overloaded, and the utilization factor of the system exceeds the maximum allowable utilization for RMS algorithm. It will display the error message in the console.

7 References

Bhat, V. (n.d.). Bhatvineeth/Schedulingsimulation: Scheduling simulator. GitHub. Retrieved March 14, 2023, from https://github.com/bhatvineeth/SchedulingSimulation

[2]Pillai, A., amp; Isha, T. (1970, January 1). A new real time simulator for task scheduling: Semantic scholar. 2012 IEEE International Conference on Computational Intelligence and Computing Research. Retrieved March 14, 2023, from https://www.semanticscholar.org/paper/A-new-real-time-simulator-for-task-scheduling-Pillai-Isha/42a6f14b8f34cd3f4b2d9bfbe65f94535a949ac4extracted

[3]React top-level API. React. (n.d.). Retrieved March 14,

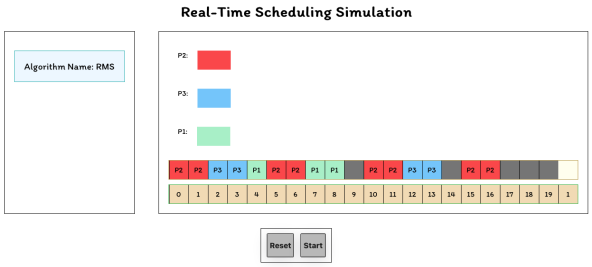

Figure 8: RMS Simulation

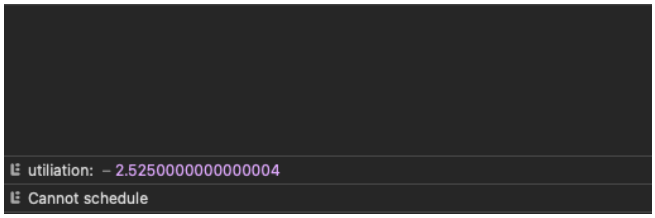

Figure 9: RMS Utilization error

2023, from https://reactjs.org/docs/react-api.html