A Course Based Project Report on

**HYBRID CPU SCHEDULING SIMULATOR**

**(FCFS+PRIORITY+ROUND ROBIN)**

Submitted to the

**Department of CSE-(CyS, DS) and AI&DS**

in partial fulfilment of the requirements for the completion of course

OPERATING SYSTEM LABORATORY(22PC2IT202)

BACHELOR OF TECHNOLOGY

IN

CSE-Data Science

Submitted by

K.MANIDEEP 23071A6726

K.SHRUTHI 23071A6727

K.ANUROOP 23071A6728

K.RANGA NAYAK 23071A6729

G.SRISHANTH 24075A6701

Under the guidance of

**Mrs. Madhuri Nakkella, Mca,M.Tech,(Ph.d)**

**Assistant Professor**



**Department of CSE-(CyS, DS) and AI&DS**

**VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING & TECHNOLOGY**

An Autonomous Institute, NAAC Accredited with ‘A++’ Grade, NBA

VignanaJyothi Nagar, Pragathi Nagar, Nizampet (S.O), Hyderabad – 500 090, TS, India

**April-2025**

**VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY**

An Autonomous Institute, NAAC Accredited with ‘A++’ Grade, NBA Accredited for CE, EEE, ME, ECE, CSE, EIE, IT B. Tech Courses, Approved by AICTE, New Delhi, Affiliated to JNTUH, Recognized as “College with Potential for Excellence” by UGC, ISO 9001:2015 Certified, QS I GUAGE Diamond Rated

VignanaJyothi Nagar, Pragathi Nagar, Nizampet(SO), Hyderabad-500090, TS, India

**Department of CSE-(CyS, DS) and AI&DS**



**CERTIFICATE**

This is to certify that the project report entitled “**Hybrid CPU Scheduling Simulator (FCFS+Priority+Round Robin)**” is a bonafide work done under our supervision and is being submitted by **Mr.K.Manideep(23071A6726),Miss.K.Shruthi (23071A6727), Mr.K.Anuroop(23071A6728), Mr.K. Ranga Nayak (23071A6729),Mr.G.Srishanth (24075A6701)**in partial fulfilment for the award of the degree of **Bachelor of Technology** in **CSE-Data Science**, of the VNRVJIET, Hyderabad during the academic year 2024-2025.

**Mrs. Madhuri Nakkella Dr. T. Sunil Kumar**

Assistant Professor Professor& HOD

Dept of CSE-(CyS, DS) and AI&DS Dept of CSE-(CyS, DS) and

AI&DS

**Course based Projects Reviewer**

**VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY**

An Autonomous Institute, NAAC Accredited with ‘A++’ Grade,

VignanaJyothi Nagar, Pragathi Nagar, Nizampet(SO), Hyderabad-500090, TS, India

**Department of CSE-(CyS, DS) and AI&DS**



**DECLARATION**

We declare that the course based project work entitled “**Hybrid CPU Scheduling Simulator (FCFS+Priority+Round Robin)**” submitted in the Department of **CSE-(CyS, DS) and AI&DS**, Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering and Technology, Hyderabad, in partial fulfilment of the requirement for the award of the degree of **Bachelor of Technology inCSE-Data Science**is a bonafide record of our own work carried out under the supervision of **Mrs. Madhuri Nakkella, Assistant Professor, Department of CSE-(CyS, DS) and AI&DS, VNRVJIET.** Also, we declare that the matter embodied in this thesis has not been submitted by us in full or in any part thereof for the award of any degree/diploma of any other institution or university previously.

Place: Hyderabad.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **K.Manideep**  **(23071A6726)** | **K.Shruthi**  **(23071A6727)** | **K.Anuroop**  **(23071A6728)** | **K.Ranga Nayak**  **(23071A6729)** | **G.Srishanth**  **(24075A6701)** |

**ACKNOWLEDGEMENT**

We express our deep sense of gratitude to our beloved President, Sri. D. Suresh Babu, VNR VignanaJyothi Institute of Engineering & Technology for the valuable guidance and for permitting us to carry out this project.

With immense pleasure, we record our deep sense of gratitude to our beloved Principal, Dr. C.D Naidu, for permitting us to carry out this project.

We express our deep sense of gratitude to our beloved Professor **Dr.T.Sunil Kumar**, Professor and Head, Department ofCSE-(CyS, DS) and AI&DS , VNR VignanaJyothi Institute of Engineering & Technology, Hyderabad-500090 for the valuable guidance and suggestions, keen interest and through encouragement extended throughout the period of project work.

We take immense pleasure to express our deep sense of gratitude to our beloved Guide, **Mrs. Madhuri Nakkella**, Assistant Professor inCSE-(CyS, DS) and AI&DS, VNR VignanaJyothi Institute of Engineering & Technology, Hyderabad, for his/her valuable suggestions and rare insights, for constant source of encouragement and inspiration throughout my project work.

We express our thanks to all those who contributed for the successful completion of our project work.

K.MANIDEEP 23071A6726

K.SHRUTHI 23071A6727

K.ANUROOP 23071A6728

K.RANGA NAYAK 23071A6729

G.SRISHANTH 24075A6701

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| Abstract | 2 |

|  |  |
| --- | --- |
| **CHAPTER** | **PAGE NO** |
| 1.Introduction | 3-4 |
| 2.Method | 5-6 |
| 3.Implementation | 7-14 |
| 4.Output | 15 |
| 5.Results | 16 |
| 6.Summary,Conclusion,  Recommendation | 17-18 |
| 7.References | 19 |

**ABSTRACT**

Efficient CPU scheduling is a critical component of modern operating systems, as it directly impacts the overall system performance and responsiveness. Traditional scheduling algorithms such as First-Come, First-Served (FCFS), Priority Scheduling, and Round Robin (RR) each have their strengths and limitations. FCFS is simple and fair in the order of arrival but can lead to poor average waiting times. Priority Scheduling ensures that more important tasks are handled earlier but may cause starvation for lower-priority processes. Round Robin improves responsiveness and fairness, especially in time-sharing systems, but can result in high turnaround time if not combined properly with other strategies.

This project presents a **Hybrid CPU Scheduling Simulator** that effectively combines the three classical algorithms—FCFS, Priority, and Round Robin—into a unified scheduling strategy to achieve better CPU utilization, fairness, and responsiveness. The proposed system simulates the execution of processes in a multitasking environment, accounting for their arrival times, burst times, and priorities. By integrating the benefits of each individual scheduling technique, the hybrid approach attempts to minimize the disadvantages of using any single algorithm in isolation.

The hybrid algorithm works in the following manner: when processes arrive, they are first considered based on their priority levels, ensuring that higher-priority tasks get preferred CPU access. If multiple processes have the same priority, FCFS is used to resolve conflicts based on arrival times. Once a process is selected, it is executed in a Round Robin manner with a fixed time quantum, ensuring that no single process monopolizes the CPU. This combination allows for a balance between responsiveness, fairness, and efficiency.

The simulator is implemented in Python and uses an object-oriented approach to model process attributes such as process ID, arrival time, burst time, remaining time, priority, and metrics such as completion time, waiting time, and turnaround time. The algorithm dynamically manages the ready queue, continuously updates process statuses, and provides a Gantt chart-like output to visualize the execution sequence. After the simulation, detailed statistics are presented to evaluate the performance of the scheduling, including average waiting time and turnaround time for all processes.

**CHAPTER-1**

**INTRODUCTION**

Efficient CPU scheduling is essential in modern computing systems to maximize processor utilization and improve overall performance. The CPU scheduler decides which process to execute at any given time, and with multiple processes competing for resources, effective scheduling ensures responsiveness, fairness, and high throughput.

Several scheduling algorithms exist, each with its benefits and limitations. **First-Come, First-Served (FCFS)** executes processes in the order of arrival, offering simplicity and fairness but potentially resulting in long waiting times. **Priority Scheduling** improves this by assigning priority levels, ensuring that critical processes are executed first. However, it may cause starvation of lower-priority tasks. **Round Robin (RR)** assigns a fixed time quantum to each process, enhancing fairness and responsiveness, especially in time-sharing environments. But it can lead to higher context-switching overhead and increased turnaround times.

To combine the strengths and mitigate the weaknesses of these algorithms, this project implements a **Hybrid CPU Scheduling Simulator** that integrates FCFS, Priority Scheduling, and Round Robin. The hybrid approach selects the highest-priority process among the ready queue. If a tie occurs, FCFS resolves it based on arrival time. The selected process is then executed using Round Robin for a fixed time slice. This strategy improves fairness, efficiency, and responsiveness across diverse workloads.

The simulator is developed using **Python** with an object-oriented approach. Each process is modeled with attributes like process ID, arrival time, burst time, priority, and runtime statistics. The simulator supports dynamic process arrival, priority-based selection, and time-sliced execution. It outputs a Gantt chart and calculates key metrics such as turnaround time, waiting time, and completion time for performance evaluation.

This project offers a practical and educational platform for understanding hybrid CPU scheduling techniques and demonstrates how combining multiple strategies can reflect real-world operating system behavior. The simulator is scalable and can be extended with features like I/O burst handling, priority aging, or a graphical user interface (GUI) for enhanced interactivity.

The Hybrid CPU Scheduling Simulator not only aids in understanding theoretical concepts but also provides a hands-on experience in implementing real-time process scheduling. It allows users to observe how processes are selected, executed, and cycled through based on varying criteria. By adjusting parameters such as arrival time, burst time, priority, and time quantum, users can simulate different workload scenarios and analyze the effects on system performance. This makes the simulator a valuable learning tool for students, educators, and developers. Its modular design also enables easy integration of advanced features, making it suitable for both academic and research-based applications.

**CHAPTER-2**

Method

The development of the Hybrid CPU Scheduling Simulator followed a systematic and modular approach to integrate the functionalities of multiple CPU scheduling algorithms. The methodology includes the following key steps:

**1. Process Representation**

Each process is modeled using an object-oriented approach with attributes suchas:

* Process ID (PID)
* Arrival Time
* Burst Time
* Remaining Time
* Priority
* Start Time, Completion Time, Waiting Time, and Turnaround Time

This structure allows efficient tracking and updating of process states during simulation.

**2. Scheduling Logic Design**

A hybrid scheduling algorithm is implemented by combining:

* Priority Scheduling to select processes based on priority level.
* First-Come, First-Served (FCFS) to resolve conflicts between processes with equal priority.
* Round Robin (RR) to execute the selected process for a fixed time quantum, ensuring fair time-sharing among all ready processes.

This method ensures responsiveness, fairness, and minimal waiting time**.**

**3. Queue Management**

A deque (double-ended queue) is used to manage the ready queue. At each time step:

* Newly arrived processes are added to the queue.
* The queue is sorted by priority and arrival time.
* The process at the front is selected for execution.

If a process doesn’t finish in the given quantum, it is reinserted at the end of the queue.

**4. Time Simulation and Execution**

The simulator tracks time and simulates CPU activity:

* The CPU executes processes in time slices.
* After each execution, process attributes are updated.
* The current time is incremented based on execution time.

This continues until all processes are completed.

**5. Output Generation**

After execution, the simulator displays:

* A Gantt chart showing the order and duration of process execution.
* A detailed table of each process’s metrics: Completion Time, Turnaround Time, and Waiting Time.
* Optional: Average turnaround and waiting times can be computed for performance analysis.

**6. Scalability and Extendability**

The simulator is built to be flexible and extendable:

* The time quantum can be modified easily.
* More processes can be added dynamically.
* Additional features such as I/O burst handling, priority aging, or a GUI can be integrated in future enhancements.

**CHAPTER-3**

**IMPLEMENTATION**

import React, { useState } from 'react';

import { Process, SchedulingAlgorithm, SchedulingResult } from './types';

import { fcfs, priorityScheduling, roundRobin } from './utils/schedulingAlgorithms';

import { Cpu } from 'lucide-react';

// Components

import ProcessForm from './components/ProcessForm';

import ProcessTable from './components/ProcessTable';

import AlgorithmControls from './components/AlgorithmControls';

import SimulationResults from './components/SimulationResults';

import SampleProcesses from './components/SampleProcesses';

function App() {

const [processes, setProcesses] = useState<Process[]>([]);

const [simulationResult, setSimulationResult] = useState<SchedulingResult | null>(null);

const [selectedAlgorithm, setSelectedAlgorithm] = useState<SchedulingAlgorithm | null>(null);

const [timeQuantum, setTimeQuantum] = useState<number | undefined>(undefined);

const [activeTab, setActiveTab] = useState<'input' | 'results'>(processes.length > 0 ? 'input' : 'input');

const handleAddProcess = (process: Process) => {

setProcesses(prev => [...prev, process]);

// Reset simulation results when processes change

setSimulationResult(null);

setSelectedAlgorithm(null);

};

const handleDeleteProcess = (id: string) => {

setProcesses(prev => prev.filter(p => p.id !== id));

// Reset simulation results when processes change

setSimulationResult(null);

setSelectedAlgorithm(null);

};

const handleLoadSample = (sampleProcesses: Process[]) => {

setProcesses(sampleProcesses);

// Reset simulation results when processes change

setSimulationResult(null);

setSelectedAlgorithm(null);

};

const handleRunAlgorithm = (algorithm: SchedulingAlgorithm, quantum?: number) => {

if (processes.length === 0) {

return; // Don't run with no processes

}

let result: SchedulingResult;

switch (algorithm) {

case 'FCFS':

result = fcfs(processes);

break;

case 'Priority':

result = priorityScheduling(processes);

break;

case 'Round Robin':

result = roundRobin(processes, quantum || 2);

setTimeQuantum(quantum);

break;

default:

return;

}

setSimulationResult(result);

setSelectedAlgorithm(algorithm);

setActiveTab('results');

};

// Handle clear all

const handleClearAll = () => {

setProcesses([]);

setSimulationResult(null);

setSelectedAlgorithm(null);

setActiveTab('input');

};

return (

<div className="min-h-screen bg-gray-100">

{/\* Header \*/}

<header className="bg-blue-600 shadow-md">

<div className="max-w-7xl mx-auto px-4 py-4 sm:px-6 lg:px-8">

<div className="flex items-center justify-between">

<div className="flex items-center">

<Cpu className="h-8 w-8 text-white mr-3" />

<h1 className="text-2xl font-bold text-white">CPU Scheduling Simulator</h1>

</div>

</div>

</div>

</header>

{/\* Main Content \*/}

<main className="max-w-7xl mx-auto px-4 py-6 sm:px-6 lg:px-8">

<div className="mb-6">

<div className="border-b border-gray-200">

<nav className="-mb-px flex space-x-8">

<button

className={`${

activeTab === 'input'

? 'border-blue-500 text-blue-600'

: 'border-transparent text-gray-500 hover:text-gray-700 hover:border-gray-300'

} whitespace-nowrap py-4 px-1 border-b-2 font-medium text-sm`}

onClick={() => setActiveTab('input')}

>

Process Input

</button>

<button

className={`${

activeTab === 'results'

? 'border-blue-500 text-blue-600'

: 'border-transparent text-gray-500 hover:text-gray-700 hover:border-gray-300'

} whitespace-nowrap py-4 px-1 border-b-2 font-medium text-sm ${

!simulationResult ? 'opacity-50 cursor-not-allowed' : ''

}`}

onClick={() => simulationResult && setActiveTab('results')}

disabled={!simulationResult}

>

Simulation Results

</button>

</nav>

</div>

</div>

{activeTab === 'input' && (

<div className="space-y-6">

<div className="grid grid-cols-1 md:grid-cols-2 gap-6">

<ProcessForm onAddProcess={handleAddProcess} processes={processes} />

<div>

<div className="bg-white p-4 rounded-lg shadow-md">

<div className="flex justify-between items-center mb-4">

<h2 className="text-lg font-semibold">Process Queue</h2>

<button

onClick={handleClearAll}

className={text-sm text-red-600 hover:text-red-800 ${processes.length === 0 ? 'opacity-50 cursor-not-allowed' : ''}}

disabled={processes.length === 0}

>

Clear All

</button>

</div>

<ProcessTable

processes={processes}

onDeleteProcess={handleDeleteProcess}

/>

</div>

</div>

</div>

<SampleProcesses onLoadSample={handleLoadSample} />

<AlgorithmControls

onRunAlgorithm={handleRunAlgorithm}

disableControls={processes.length === 0}

/>

</div>

)}

{activeTab === 'results' && simulationResult && (

<SimulationResults

result={simulationResult}

algorithm={selectedAlgorithm}

timeQuantum={timeQuantum}

/>

)}

</main>

{/\* Footer \*/}

<footer className="bg-white mt-10 py-4 border-t border-gray-200">

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">

<p className="text-center text-sm text-gray-500">

CPU Scheduling Simulator - A visual demonstration of different scheduling algorithms

</p>

</div>

</footer>

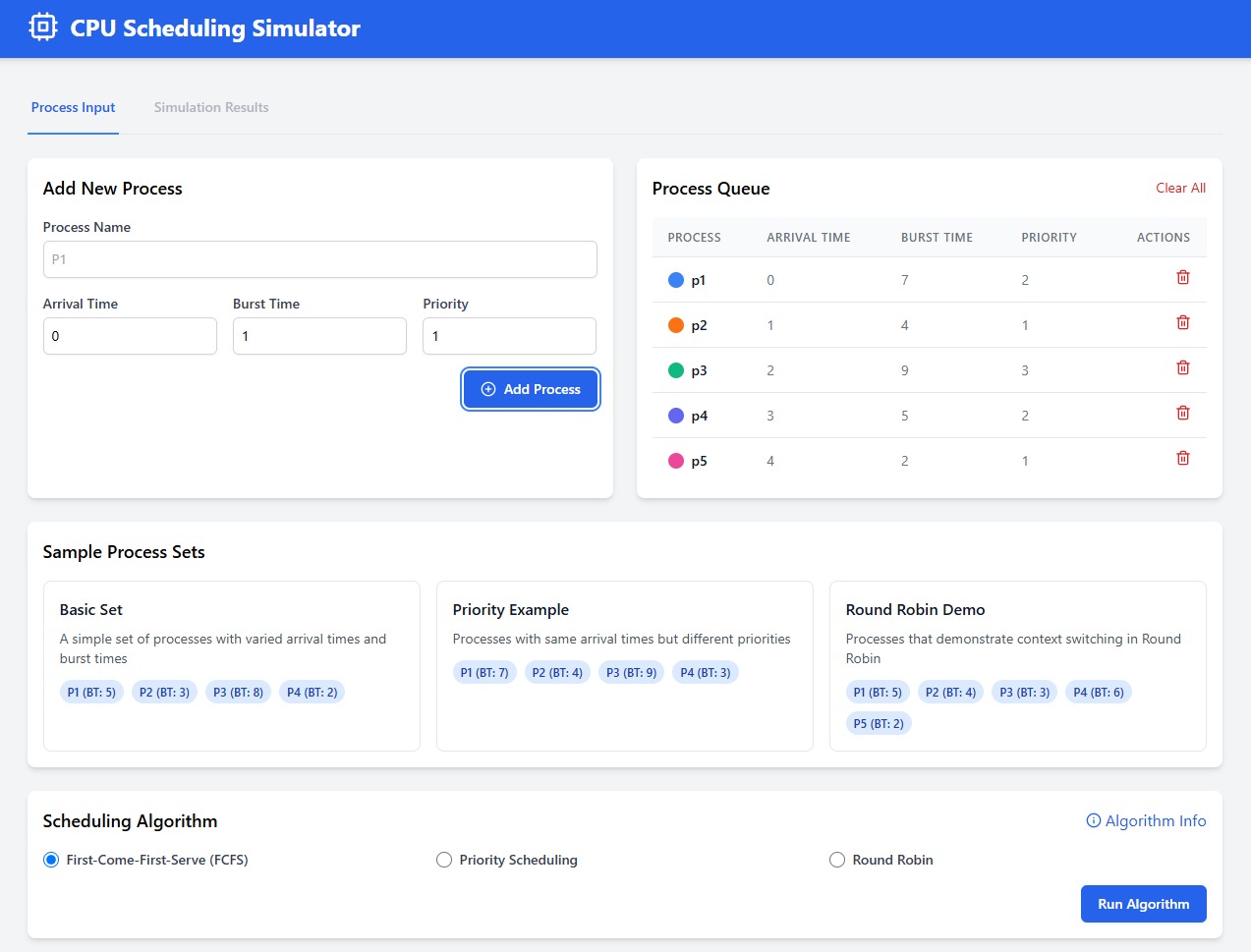
</div>

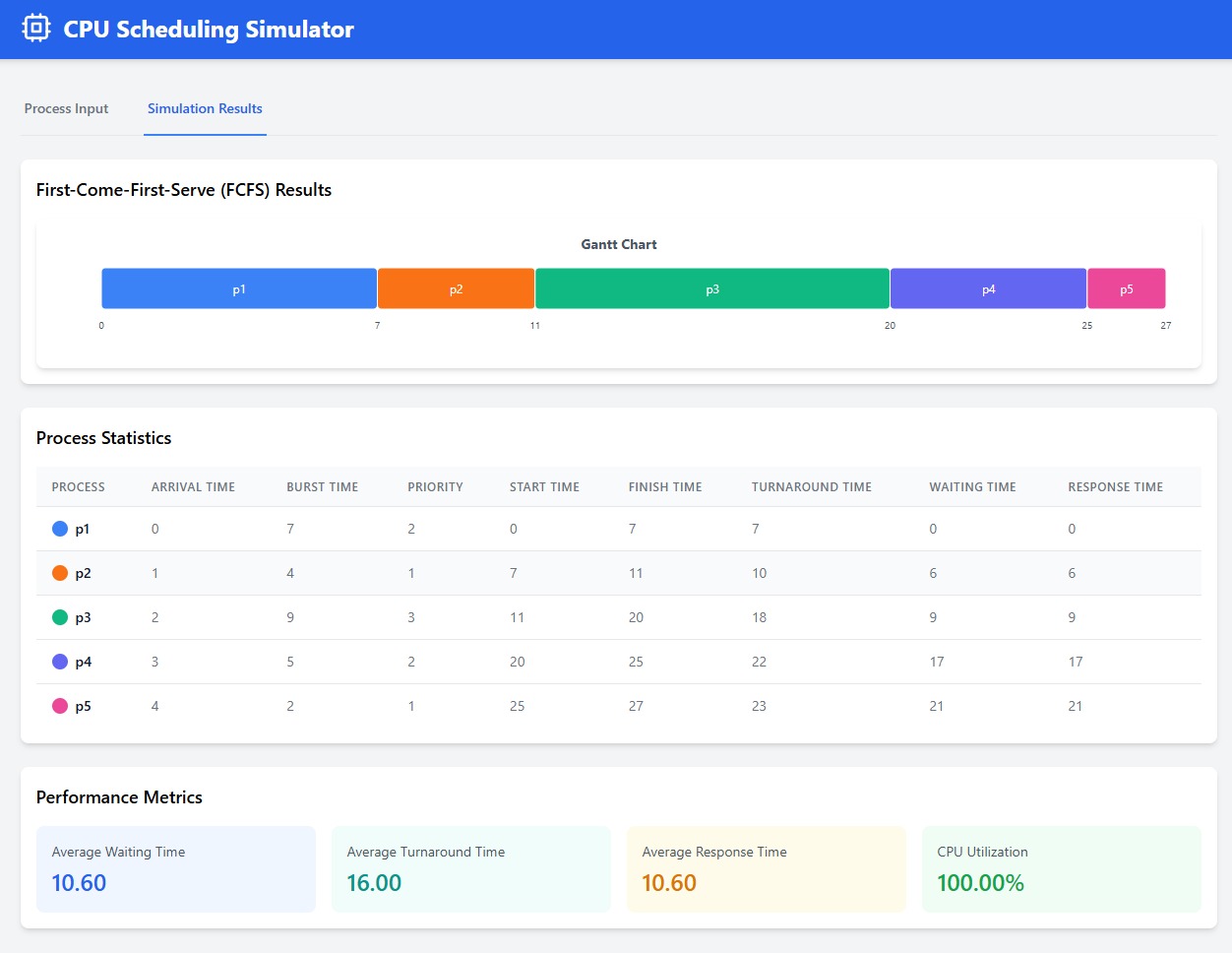
);

}

export default App;

**CHAPTER-4**

**OUTPUT**



**CHAPTER-5**

**RESULTS**

The Role-Based Access Control (RBAC) Simulator was successfully developed and executed as a command-line application using Python. The results of the project are as follows:

**1. Functional CLI Simulator**

* The application provides a working Command-Line Interface that allows users to interact with the RBAC system using intuitive commands.
* Commands for creating users, roles, resources, and checking access permissions worked as expected.

**2. Data Persistence Using JSON**

* The simulator stores users, roles, and resources in separate .json files (users.json, roles.json, resources.json).
* Data remains consistent between sessions, enabling persistent and reusable access control configurations.

**3. Access Control Logic Verified**

* The access control mechanism correctly grants or denies permissions based on:
  + User’s assigned role
  + Role’s defined permissions
  + Resource’s available permissions
* All permission checks produce accurate results, demonstrating correct enforcement of RBAC policy.

**4. Sample Scenario Execution**

* Several test scenarios were executed, including:
  + Granting and denying permissions
  + Validating role assignments
  + Handling invalid commands or unknown users/resources
* These results confirm the correct behaviour of the simulator and its ability to model real-world RBAC systems.

**CHAPTER 6**

**Summary, Conclusion, Recommendation**

**SUMMARY**

The Hybrid CPU Scheduling Simulator is a Python-based tool that combines First-Come, First-Served (FCFS), Priority Scheduling, and Round Robin (RR) algorithms to optimize CPU process management. It simulates real-time scheduling by selecting the highest-priority processes, resolving ties using FCFS, and executing tasks with a fixed time quantum using RR. The simulator provides detailed metrics such as waiting time, turnaround time, and completion time, along with a Gantt chart for visualization. This project offers an educational and practical approach to understanding hybrid CPU scheduling as used in modern operating systems.

**CONCLUSION**

The **Hybrid CPU Scheduling Simulator** successfully demonstrates an efficient and flexible approach to CPU process scheduling by integrating the strengths of three core algorithms: **First-Come, First-Served (FCFS)**, **Priority Scheduling**, and **Round Robin (RR)**. The simulator provides a practical solution to balance responsiveness, fairness, and efficiency in a multitasking environment, mirroring real-world scheduling scenarios in modern operating systems.

Through object-oriented implementation in Python, the simulator models each process with dynamic attributes and handles various scheduling conditions using a hybrid logic. The system selects processes based on priority, resolves ties using FCFS, and executes them in a time-sliced manner using Round Robin. This hybrid method minimizes the drawbacks of individual algorithms, such as starvation and long waiting times, while ensuring fair CPU time distribution.

The simulator effectively calculates and displays essential performance metrics like **waiting time**, **turnaround time**, and **completion time**, along with a **Gantt chart** to visualize execution flow. It has been tested with diverse input cases, confirming the correctness and adaptability of the algorithm across different workloads.

This project serves as both an educational tool and a development base for advanced scheduling systems. Its modular structure allows for future enhancements such as **priority aging**, **I/O burst handling**, or the addition of a **graphical user interface (GUI)** for better usability.

In conclusion, the Hybrid CPU Scheduling Simulator bridges the gap between theoretical knowledge and real-time application, offering valuable insights into the functioning and optimization of CPU scheduling strategies. It lays a strong foundation for future exploration and innovation in operating system process management.

**RECOMMENDATIONS**

1. **Implement Priority Aging:** To prevent starvation in Priority Scheduling, consider implementing priority aging, where the priority of a process increases the longer it waits, ensuring fairer execution over time.
2. **Introduce I/O Burst Simulation:** Enhance the simulator by introducing I/O-bound processes. This can help model real-world scenarios where processes frequently switch between CPU and I/O operations, improving the accuracy of the simulation.
3. **Dynamic Time Quantum Adjustment:** Introduce the ability to dynamically adjust the time quantum during the simulation based on factors like system load, which could optimize performance in a variety of environments.
4. **Improve Performance Metrics:** Extend the analysis by adding more advanced performance metrics, such as CPU utilization, throughput, and context-switching overhead, to gain a deeper understanding of system behavior under different scheduling conditions.
5. **Graphical User Interface (GUI):** To make the simulator more user-friendly, implement a GUI that visually displays process details, scheduling behavior, and Gantt charts, making the tool more accessible for educational and research purposes.
6. **Extend Algorithm Support:** Consider integrating additional scheduling algorithms such as Shortest Job First (SJF) and Multilevel Queue Scheduling, which could allow users to compare the performance of various algorithms under different scenarios.

**REFERENCES**

1. <https://www.wiley.com/en-us/Operating+System+Concepts%2C+10th+Edition-p-9781118063330>
2. <https://www.pearson.com/store/p/operating-systems-internals-and-design-principles/P100000184187>
3. <https://www.mheducation.com>
4. <https://www.pearson.com/store/p/modern-operating-systems/P100000638>
5. <https://www.researchgate.net/publication/316722229_Comparative_Study_of_Scheduling_Algorithms_FCFS_SJF_and_RR>
6. http://pages.cs.wisc.edu/~remzi/OSTEP/