	Sanjivani Rural Educational Society's **SANJIVANI COLLEGE OF ENGINEERING** **(An Autonomous Institution)** Kopargaon – 423 603, Maharashtra.	**ACAD-F-15 K**
Academic Year: 2024-25	**CIA ACTIVITY REPORT**	**Revision : 00** **Dated : _____**
Department :	**Computer Engineering**	**Date of Preparation : _____**
Course Code & Name:	**System Software**	**Year/Sem: TY/ VI**

Activity Title:

Process Scheduling Simulator

Submitted by:

Roll No	PRN No	Name of the Student	Signature
1	**UCS22M1001**	**Achare Tanmay**	
2	**UCS22M1125**	**Tambe Ajinkya**	
3	**UCS22F1137**	**Walke Akshata**	
4	**UCS22F1002**	**Unde Ashwini**	

Course Incharge :

Dr.S.N.Gunjal

Report Index

1.Introduction

Process scheduling is a key function of modern operating systems that ensures efficient use of the CPU and smooth multitasking. It determines the order in which processes are executed, helping manage both time and system resources effectively. The Process Scheduling Simulator is a software tool designed to model and demonstrate how different scheduling algorithms work. By simulating these algorithms, users can observe how the CPU handles multiple processes and how this impacts key performance metrics such as waiting time, turnaround time, and CPU idle time.

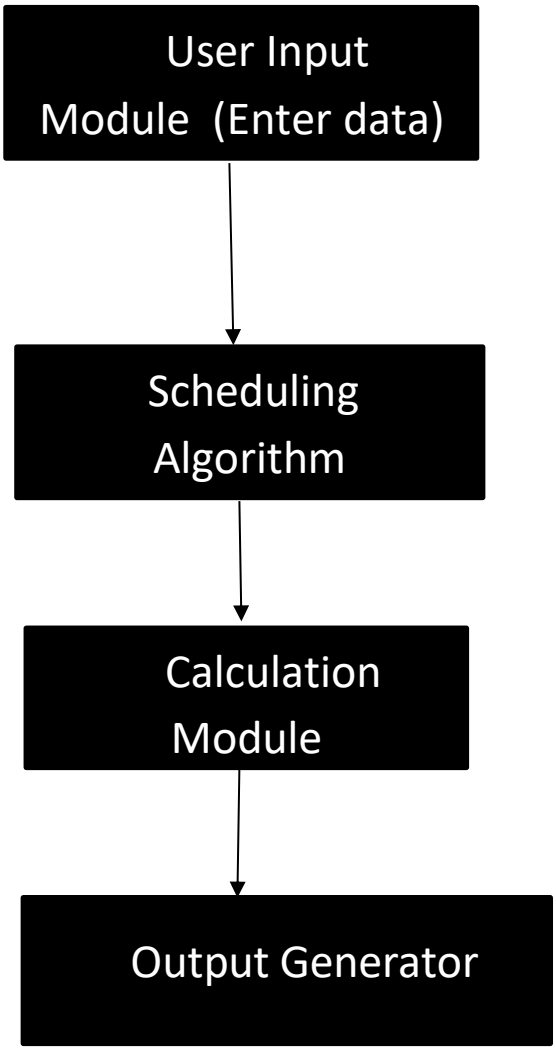
This simulator focuses on four widely used scheduling techniques: First-Come, First-Served (FCFS), Shortest Job Next (SJN), Round Robin (RR), and Priority Scheduling. Each of these algorithms uses a unique method to decide which process should run next. The simulator visually compares the behavior and output of these algorithms, making it easier to understand their strengths and weaknesses in various scenarios.

The main goal of this project is to provide a user-friendly, educational platform for students and learners to explore CPU scheduling. It allows users to enter process data, run simulations, and analyze results using Gantt charts and metric tables. This interactive approach helps bridge the gap between theoretical learning and practical application. By working with this simulator, users not only deepen their understanding of operating system concepts but also enhance their logical thinking, coding, and analytical skills.

2.Objective

The main objective of the Process Scheduling Simulator project is to provide an interactive and educational tool for understanding how various CPU scheduling algorithms function in an operating system environment. This simulator aims to help users, especially students, visualize and analyze the execution of processes under different scheduling techniques such as First-Come, First-Served (FCFS), Shortest Job Next (SJN), Round Robin (RR), and Priority Scheduling. By allowing users to input process data and select a scheduling method, the simulator calculates and displays important performance metrics like waiting time, turnaround time, and response time. It also generates Gantt charts to illustrate the process flow. The tool is designed to make complex scheduling concepts easier to grasp, enhance problem-solving skills, and strengthen the practical understanding of how operating systems manage multiple tasks. Overall, the project encourages active learning and bridges the gap between theoretical knowledge and real-world application.

3.Block Diagram



4.Explanation of Each Module

The Process Scheduling Simulator is divided into several core modules, each responsible for handling a specific task within the simulation process. These modules work together to provide a smooth and interactive user experience while simulating CPU scheduling algorithms.

1. User Input Module

The User Input Module serves as the interface for the user to provide the necessary information for the simulation. It takes in:

- Number of Processes: The user specifies how many processes are to be scheduled.
- Process Details: For each process, the user inputs:
    - Process ID: A unique identifier (e.g., P1, P2, etc.).
    - Arrival Time: The time at which the process enters the ready queue.
    - Burst Time: The amount of CPU time required to execute the process.
    - Priority: If the user selects a priority-based algorithm, the priority for each process is also entered.

This module validates that the inputs are correct (non-negative values, proper format) and ensures that no data is missing. Once validated, the data is forwarded to the Scheduling Algorithm Module for processing.

2. Scheduling Algorithm Module

The Scheduling Algorithm Module is responsible for applying the selected scheduling algorithm to decide the order in which processes are executed. It takes the user input (process details) and follows the logic of the chosen scheduling method to allocate CPU time to processes. The available algorithms are:

- First-Come, First-Served (FCFS): The first process to arrive gets executed first. This method is simple but can result in high waiting times for long processes.
- Shortest Job Next (SJN): Also called Shortest Job First (SJF), this algorithm selects the process with the shortest burst time for execution next, reducing overall waiting time but potentially causing starvation for longer processes.

- Round Robin (RR): A preemptive scheduling algorithm where each process is given a fixed time slice (quantum). If the process doesn't finish in its allotted time, it’s placed back in the ready queue for the next round.
- Priority Scheduling: Processes are assigned priorities, and the CPU executes the highest-priority process first. It can be either preemptive or non-preemptive, depending on whether the running process can be interrupted.

The module processes the data based on the algorithm and prepares a sequence of process executions that is passed to the Calculation Module for performance measurement.

3. Calculation Module

The Calculation Module computes several important metrics that evaluate the performance of the chosen scheduling algorithm. These metrics help in understanding the efficiency of the process scheduling. The key metrics calculated are:

- Completion Time: The time when the process finishes execution. It is the final time of the process in the simulation.
- Waiting Time: The amount of time a process spends in the ready queue waiting to be executed. It is calculated as:
- Turnaround Time: The total time taken from the arrival of the process to its completion, including both waiting and execution times. It is calculated as:

Additionally, if applicable, the Response Time (the time between when the process arrives and when it starts execution) can be calculated. After computing these metrics for each process, the module calculates average waiting time, average turnaround time, and possibly CPU utilization to give a summary of the scheduling algorithm’s performance.

4. Output Generator Module

The Output Generator Module displays the results of the scheduling simulation:

- Tabular Output: A table showing:
    - Process ID: Unique identifier for each process.
    - Arrival Time: When the process arrives in the ready queue.
    - Burst Time: Time needed for execution.

- Completion Time: Time when the process finishes.
- Waiting Time: Time the process waits in the ready queue.
- Turnaround Time: Total time from arrival to completion.

Gantt Chart: A visual chart showing when each process is executed, with each bar representing a process's burst time.

Average Metrics: Displays the average waiting time and average turnaround time for all processes.

5.Working

1. User Input:
The user starts by providing details for the simulation, including the number of processes, their arrival times, burst times, and priorities (if applicable). The input is gathered through a user interface and validated to ensure the data is correct.

2.Scheduling Algorithm Selection:
The user selects one of the available scheduling algorithms:

- FCFS (First-Come, First-Served): Processes are executed in the order they arrive.
- SJN (Shortest Job Next): The process with the shortest burst time is executed next.
- Round Robin (RR): Each process gets a fixed time slice for execution, and if it doesn't finish, it is put back in the queue.
- Priority Scheduling: The process with the highest priority is executed first.

3. Execution Simulation:
Based on the selected algorithm, the simulator organizes the processes and determines the order in which they are executed. For example:

- FCFS executes processes in the order they arrive.
- SJN chooses the process with the smallest burst time, and so on.

4. Metric Calculation:
As processes are executed, the simulator calculates key performance metrics:

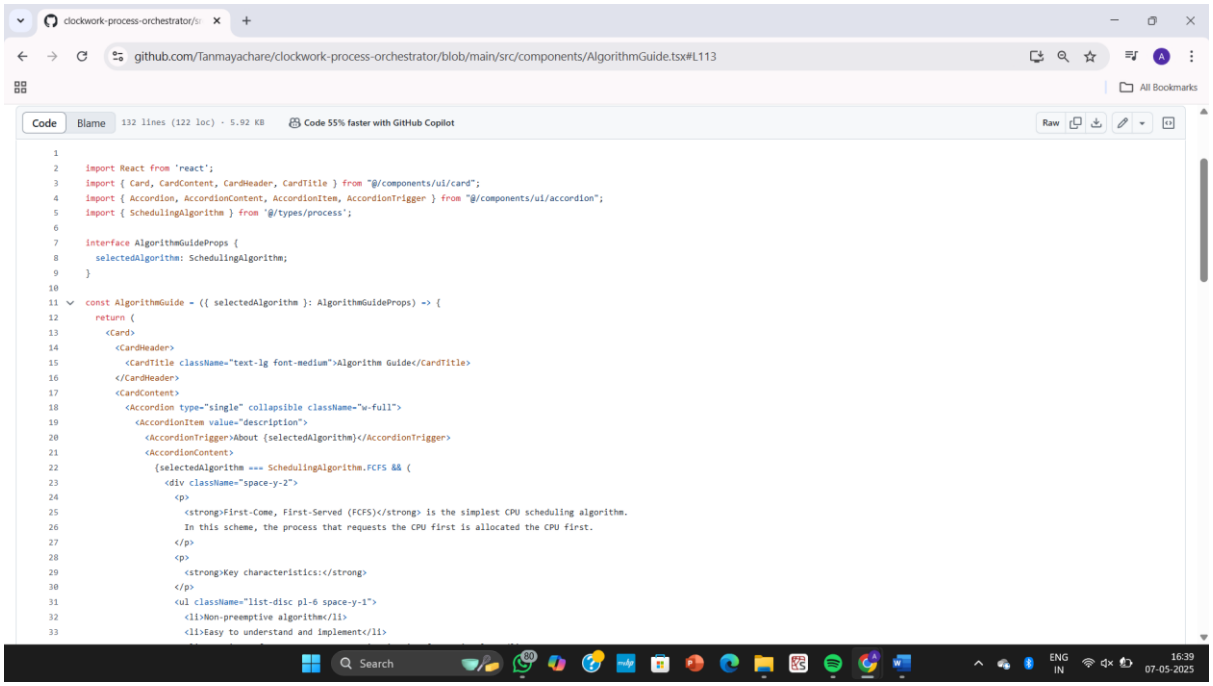
- Completion Time: When a process finishes execution.
- Waiting Time: Time spent waiting in the queue.
- Turnaround Time: The total time a process spends in the system from arrival to completion.

5. Results Display:
After all processes have been executed:

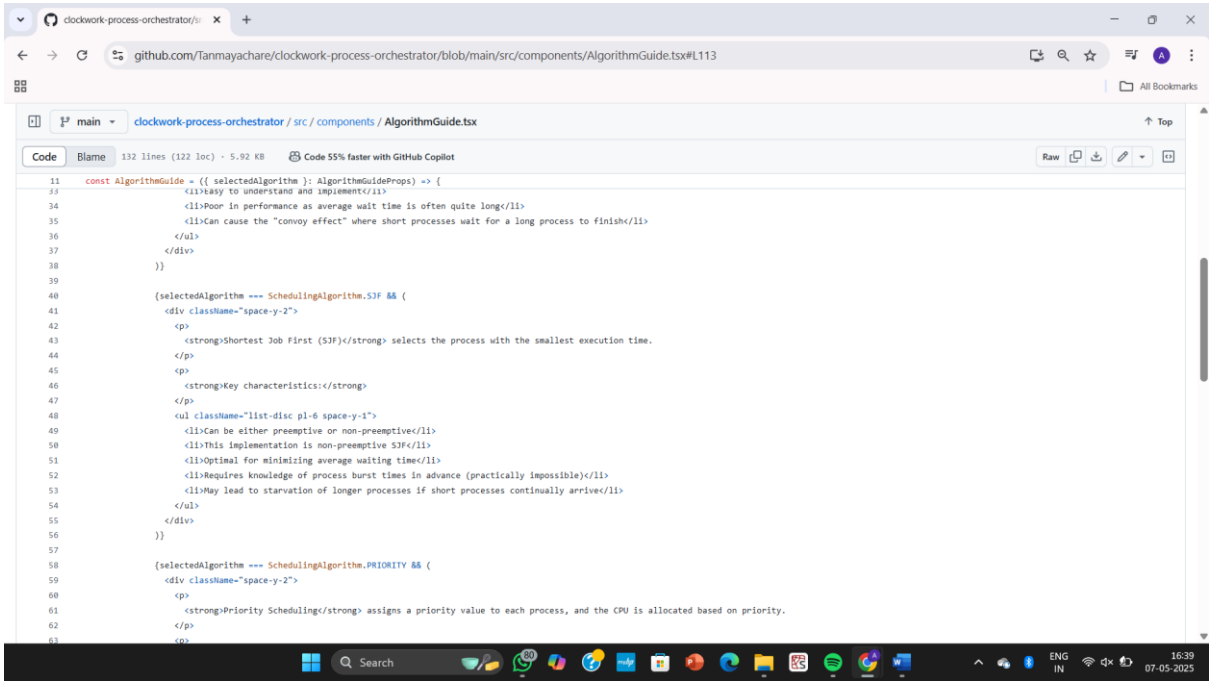
- The Tabular Output displays a table with the calculated metrics for each process.
- The Gantt Chart visually shows the execution timeline of each process.
- Average Metrics: The average waiting time and turnaround time are calculated for all processes.

6.Visualization and Analysis:
The simulator provides both visual (Gantt chart) and numerical results (table), helping the user understand the impact of different scheduling algorithms on process execution. This makes it easier to compare the efficiency of each algorithm.

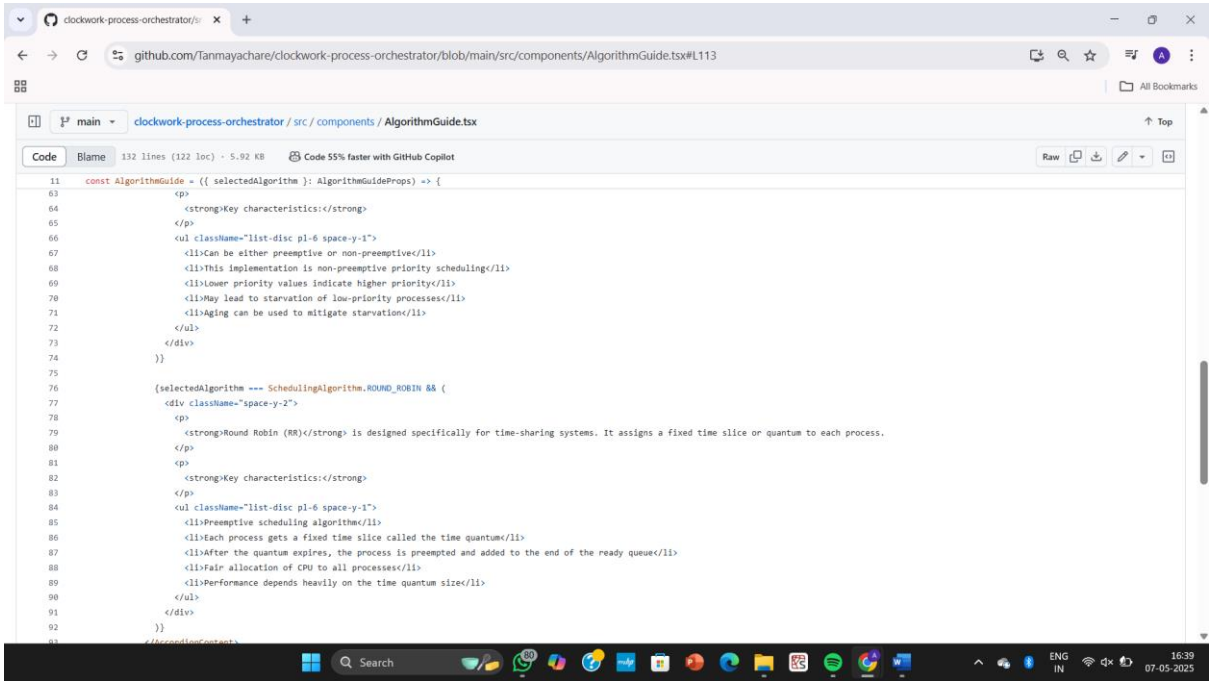
6.Code



```
import React from 'react';
import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card";
import { Accordion, AccordionContent, AccordionItem, AccordionTrigger } from "@/components/ui/accordion";
import { SchedulingAlgorithm } from '@/types/process';

interface AlgorithmGuideProps {
  selectedAlgorithm: SchedulingAlgorithm;
}

const AlgorithmGuide = ({ selectedAlgorithm }: AlgorithmGuideProps) => {
  return (
    <Card>
      <CardHeader>
        <CardTitle className="text-lg font-medium">Algorithm Guide</CardTitle>
      </CardHeader>
      <CardContent>
        <Accordion type="single" collapsible className="w-full">
          <AccordionItem value="description">
            <AccordionTrigger>About {selectedAlgorithm}</AccordionTrigger>
            <AccordionContent>
              {selectedAlgorithm === SchedulingAlgorithm.FCFS && (
                <div className="space-y-2">
                  <p>
                    <strong>First-Come, First-Served (FCFS)</strong> is the simplest CPU scheduling algorithm.
                    In this scheme, the process that requests the CPU first is allocated the CPU first.
                  </p>
                  <p>
                    <strong>Key characteristics:</strong>
                  </p>
                  <ul className="list-disc pl-6 space-y-1">
                    <li>Non-preemptive algorithm</li>
                    <li>Easy to understand and implement</li>
```



```
const AlgorithmGuide = ({ selectedAlgorithm }: AlgorithmGuideProps) => {
                    <li>Easy to understand and implement</li>
                    <li>Poor in performance as average wait time is often quite long</li>
                    <li>Can cause the "convoy effect" where short processes wait for a long process to finish</li>
                  </ul>
                </div>
              )}

              {selectedAlgorithm === SchedulingAlgorithm.SJF && (
                <div className="space-y-2">
                  <p>
                    <strong>Shortest Job First (SJF)</strong> selects the process with the smallest execution time.
                  </p>
                  <p>
                    <strong>Key characteristics:</strong>
                  </p>
                  <ul className="list-disc pl-6 space-y-1">
                    <li>Can be either preemptive or non-preemptive</li>
                    <li>This implementation is non-preemptive SJF</li>
                    <li>Optimal for minimizing average waiting time</li>
                    <li>Requires knowledge of process burst times in advance (practically impossible)</li>
                    <li>May lead to starvation of longer processes if short processes continually arrive</li>
                  </ul>
                </div>
              )}

              {selectedAlgorithm === SchedulingAlgorithm.PRIORITY && (
                <div className="space-y-2">
                  <p>
                    <strong>Priority Scheduling</strong> assigns a priority value to each process, and the CPU is allocated based on priority.
                  </p>
                  <p>
```


clockwork-process-orchestrator/s
github.com/Tanmayachare/clockwork-process-orchestrator/blob/main/src/components/AlgorithmGuide.tsx#L113
All Bookmarks
main
clockwork-process-orchestrator / src / components / AlgorithmGuide.tsx
Top
Code
Blame
132 lines (122 loc) · 5.92 KB
Code 55% faster with GitHub Copilot
Raw
const AlgorithmGuide = ({ selectedAlgorithm }: AlgorithmGuideProps) => {

Key characteristics:

<ul className="list-disc pl-6 space-y-1">
Can be either preemptive or non-preemptive
This implementation is non-preemptive priority scheduling
Lower priority values indicate higher priority
May lead to starvation of low-priority processes
Aging can be used to mitigate starvation

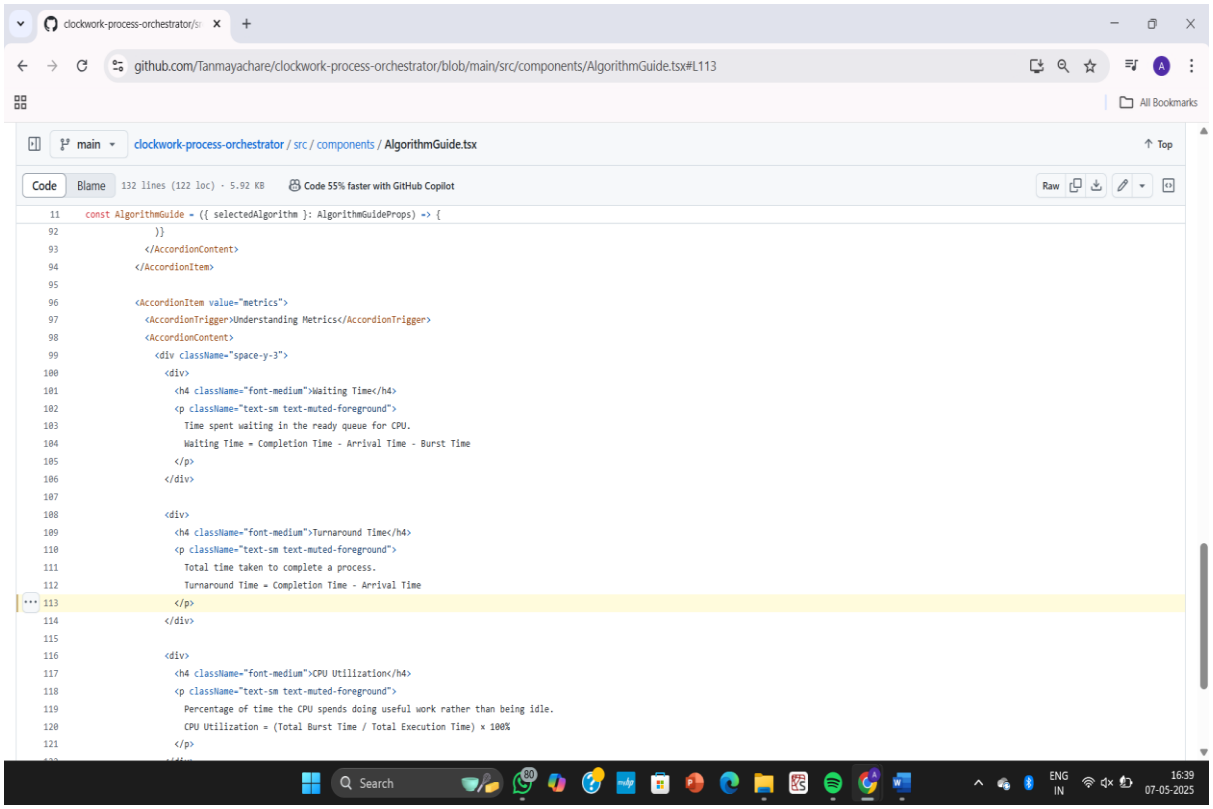
)}
{selectedAlgorithm === SchedulingAlgorithm.ROUND_ROBIN && (

Round Robin (RR) is designed specifically for time-sharing systems. It assigns a fixed time slice or quantum to each process.

Key characteristics:

<ul className="list-disc pl-6 space-y-1">
Preemptive scheduling algorithm
Each process gets a fixed time slice called the time quantum
After the quantum expires, the process is preempted and added to the end of the ready queue
Fair allocation of CPU to all processes
Performance depends heavily on the time quantum size

)}
Search
ENG
IN
16:39
07-05-2025


clockwork-process-orchestrator/s
github.com/Tanmayachare/clockwork-process-orchestrator/blob/main/src/components/AlgorithmGuide.tsx#L113
All Bookmarks
main
clockwork-process-orchestrator / src / components / AlgorithmGuide.tsx
Top
Code
Blame
132 lines (122 loc) · 5.92 KB
Code 55% faster with GitHub Copilot
Raw
const AlgorithmGuide = ({ selectedAlgorithm }: AlgorithmGuideProps) => {
)}
</AccordionContent>
</AccordionItem>
<AccordionItem value="metrics">
<AccordionTrigger>Understanding Metrics</AccordionTrigger>
<AccordionContent>

<h4 className="font-medium">Waiting Time</h4>

Time spent waiting in the ready queue for CPU.
Waiting Time = Completion Time - Arrival Time - Burst Time

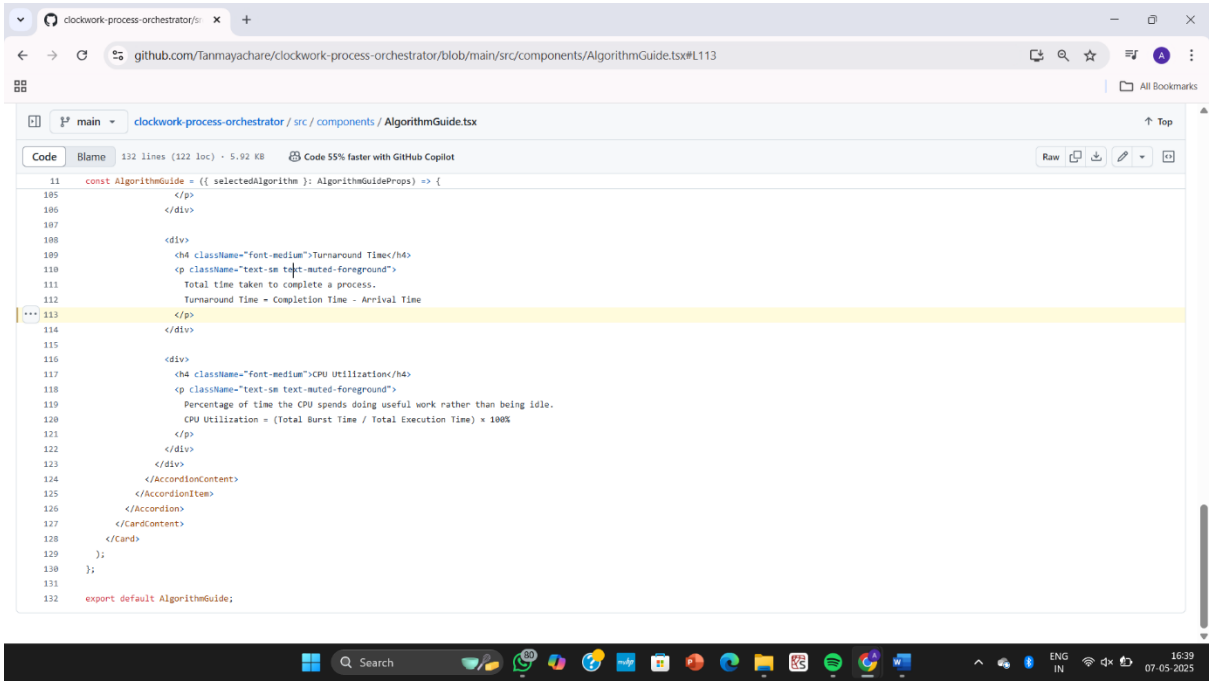
<h4 className="font-medium">Turnaround Time</h4>

Total time taken to complete a process.
Turnaround Time = Completion Time - Arrival Time

<h4 className="font-medium">CPU Utilization</h4>

Percentage of time the CPU spends doing useful work rather than being idle.
CPU Utilization = (Total Burst Time / Total Execution Time) x 100%

Search
ENG
IN
16:39
07-05-2025


github.com/Tanmayachare/clockwork-process-orchestrator/blob/main/src/components/AlgorithmGuide.tsx#L113
main
clockwork-process-orchestrator / src / components / AlgorithmGuide.tsx
Code
Blame
132 lines (122 loc) · 5.92 KB
Code 55% faster with GitHub Copilot
Raw
const AlgorithmGuide = ({ selectedAlgorithm }: AlgorithmGuideProps) => {

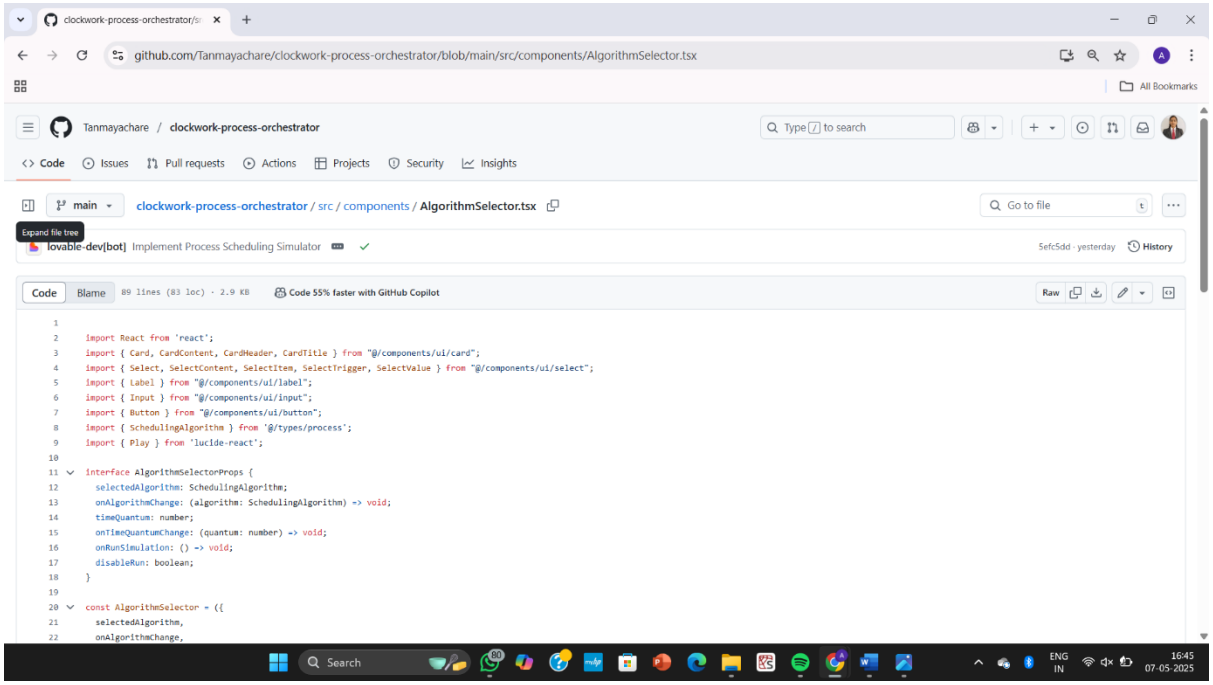
<h4 className="font-medium">Turnaround Time</h4>

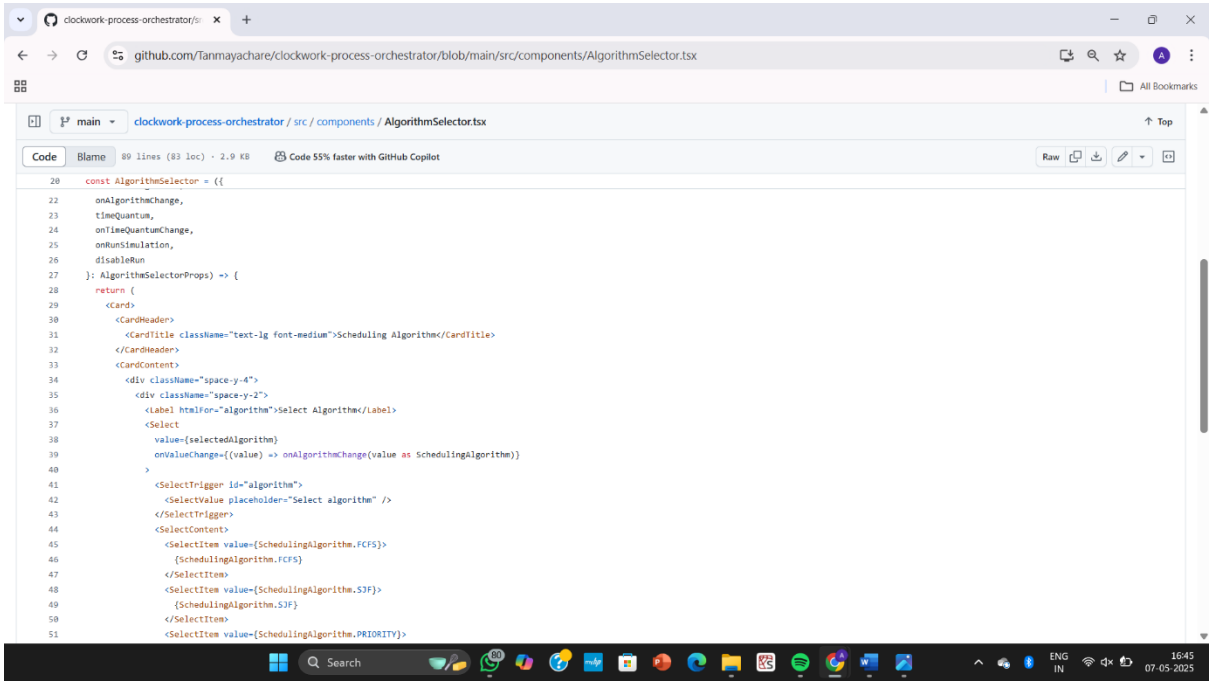
Total time taken to complete a process.
Turnaround Time = Completion Time - Arrival Time

<h4 className="font-medium">CPU Utilization</h4>

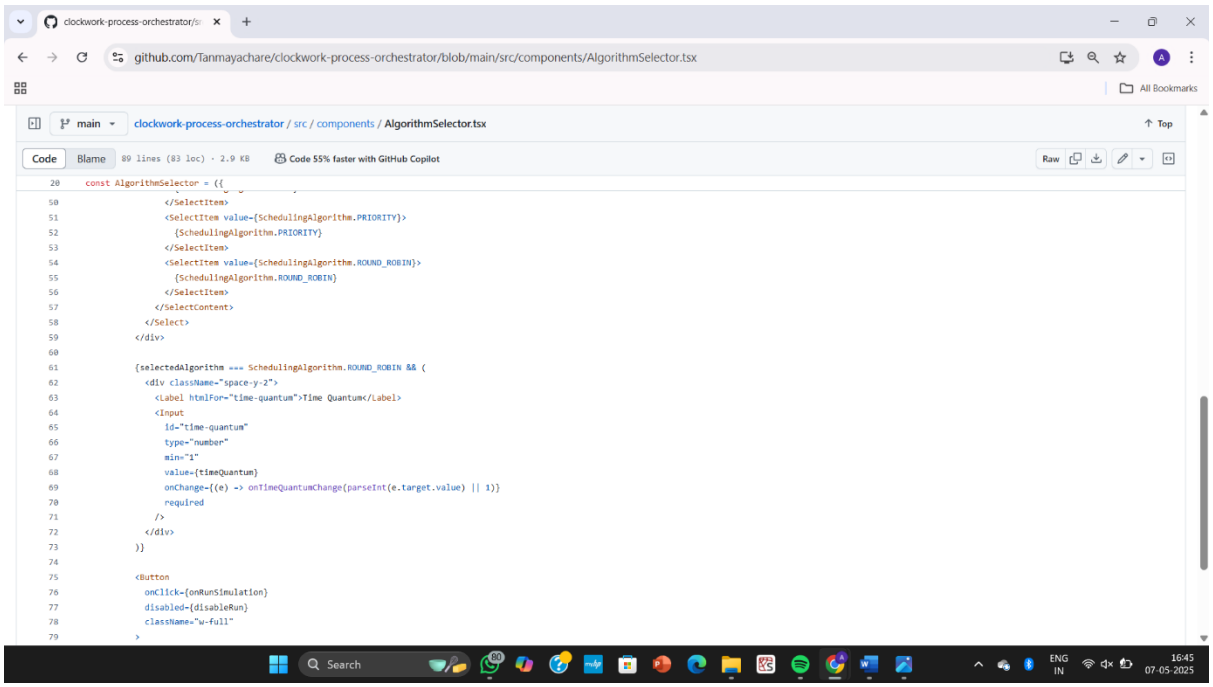
Percentage of time the CPU spends doing useful work rather than being idle.
CPU Utilization = (Total Burst Time / Total Execution Time) x 100%

</AccordionContent>
</AccordionItem>
</Accordion>
</CardContent>
</Card>
);
};
export default AlgorithmGuide;
Search
ENG
IN
16:39
07-05-2025


github.com/Tanmayachare/clockwork-process-orchestrator/blob/main/src/components/AlgorithmSelector.tsx
Tanmayachare / clockwork-process-orchestrator
Code
Issues
Pull requests
Actions
Projects
Security
Insights
main
clockwork-process-orchestrator / src / components / AlgorithmSelector.tsx
Expand file tree
lovable-dev[bot] Implement Process Scheduling Simulator
5efc5dd · yesterday
History
Code
Blame
89 lines (83 loc) · 2.9 KB
Code 55% faster with GitHub Copilot
Raw
import React from 'react';
import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card";
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from "@/components/ui/select";
import { Label } from "@/components/ui/label";
import { Input } from "@/components/ui/input";
import { Button } from "@/components/ui/button";
import { SchedulingAlgorithm } from '@/types/process';
import { Play } from 'lucide-react';
interface AlgorithmSelectorProps {
selectedAlgorithm: SchedulingAlgorithm;
onAlgorithmChange: (algorithm: SchedulingAlgorithm) => void;
timeQuantum: number;
onTimeQuantumChange: (quantum: number) => void;
onRunSimulation: () => void;
disableRun: boolean;
}
const AlgorithmSelector = ({
selectedAlgorithm,
onAlgorithmChange,
Search
ENG
IN
16:45
07-05-2025


clockwork-process-orchestrator/sr
github.com/Tanmayachare/clockwork-process-orchestrator/blob/main/src/components/AlgorithmSelector.tsx
All Bookmarks
main
clockwork-process-orchestrator / src / components / AlgorithmSelector.tsx
Top
Code
Blame
89 lines (83 loc) · 2.9 KB
Code 55% faster with GitHub Copilot
Raw
const AlgorithmSelector = ({
onAlgorithmChange,
timeQuantum,
onTimeQuantumChange,
onRunSimulation,
disableRun
}: AlgorithmSelectorProps) => {
return (
<Card>
<CardHeader>
<CardTitle className="text-lg font-medium">Scheduling Algorithm</CardTitle>
</CardHeader>
<CardContent>

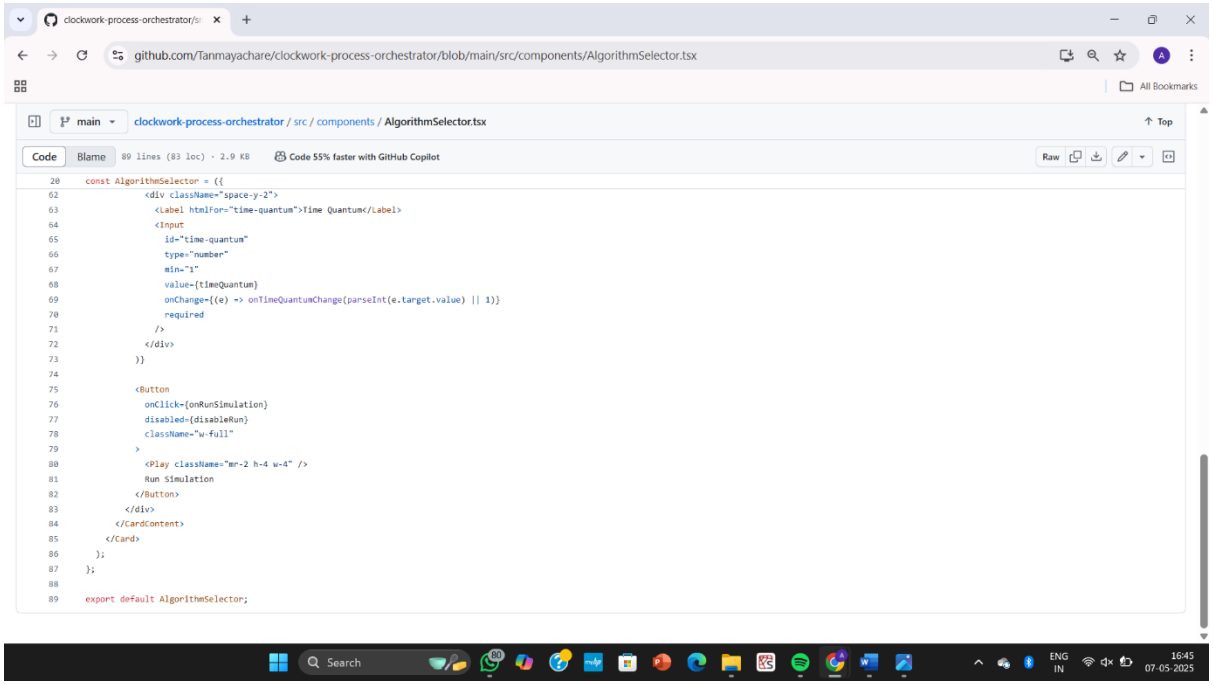
<Label htmlFor="algorithm">Select Algorithm</Label>
<Select
value={selectedAlgorithm}
onValueChange={(value) => onAlgorithmChange(value as SchedulingAlgorithm)}
>
<SelectTrigger id="algorithm">
<SelectValue placeholder="Select algorithm" />
</SelectTrigger>
<SelectContent>
<SelectItem value={SchedulingAlgorithm.FCFS}>
{SchedulingAlgorithm.FCFS}
</SelectItem>
<SelectItem value={SchedulingAlgorithm.SJF}>
{SchedulingAlgorithm.SJF}
</SelectItem>
<SelectItem value={SchedulingAlgorithm.PRIORITY}>
Search
ENG
IN
16:45
07-05-2025


clockwork-process-orchestrator/sr
github.com/Tanmayachare/clockwork-process-orchestrator/blob/main/src/components/AlgorithmSelector.tsx
All Bookmarks
main
clockwork-process-orchestrator / src / components / AlgorithmSelector.tsx
Top
Code
Blame
89 lines (83 loc) · 2.9 KB
Code 55% faster with GitHub Copilot
Raw
const AlgorithmSelector = ({
</SelectItem>
<SelectItem value={SchedulingAlgorithm.PRIORITY}>
{SchedulingAlgorithm.PRIORITY}
</SelectItem>
<SelectItem value={SchedulingAlgorithm.ROUND_ROBIN}>
{SchedulingAlgorithm.ROUND_ROBIN}
</SelectItem>
</SelectContent>
</Select>

{selectedAlgorithm === SchedulingAlgorithm.ROUND_ROBIN && (

<Label htmlFor="time-quantum">Time Quantum</Label>
<Input
id="time-quantum"
type="number"
min="1"
value={timeQuantum}
onChange={(e) => onTimeQuantumChange(parseInt(e.target.value) || 1)}
required
/>

)}
<Button
onClick={onRunSimulation}
disabled={disableRun}
className="w-full"
>
Search
ENG
IN
16:45
07-05-2025

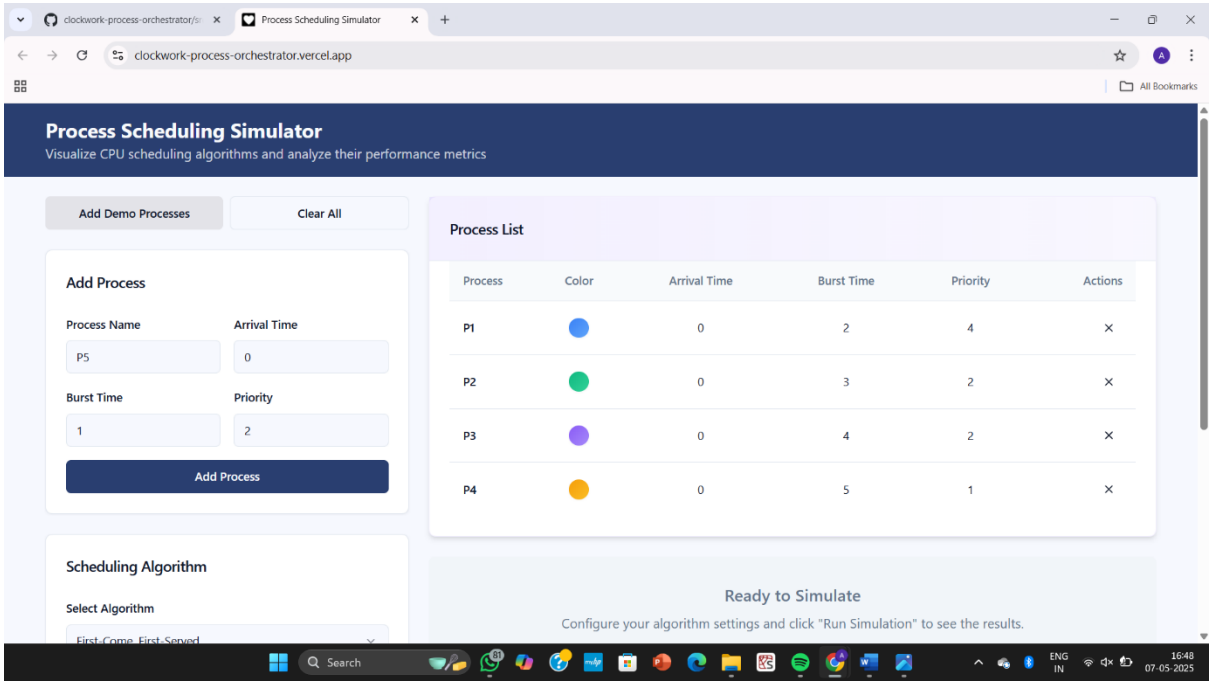

clockwork-process-orchestrator/sr
github.com/Tanmayachare/clockwork-process-orchestrator/blob/main/src/components/AlgorithmSelector.tsx
All Bookmarks
main
clockwork-process-orchestrator / src / components / AlgorithmSelector.tsx
Top
Code
Blame
89 lines (83 loc) · 2.9 KB
Code 55% faster with GitHub Copilot
Raw
const AlgorithmSelector = ({

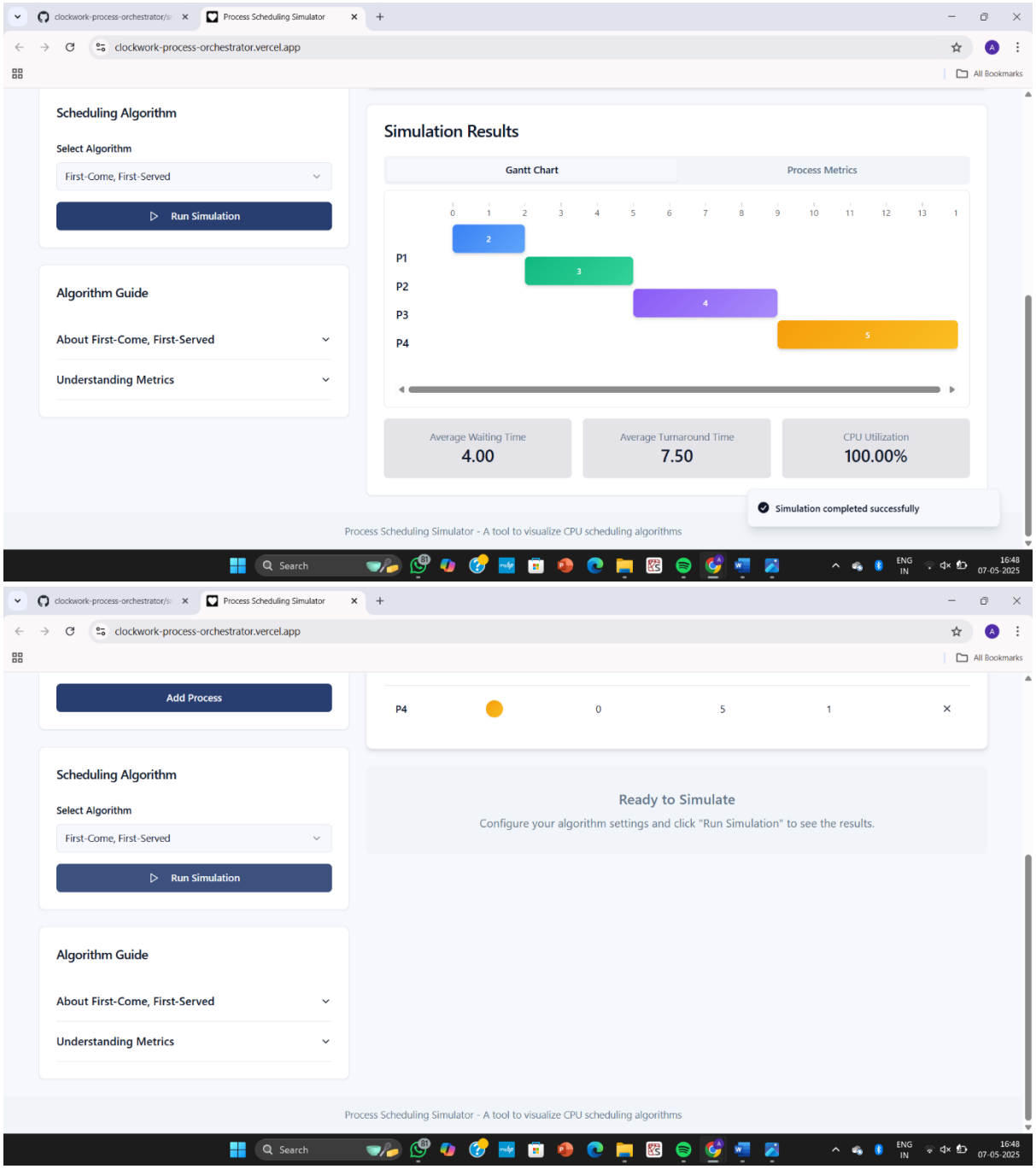
<Label htmlFor="time-quantum">Time Quantum</Label>
<Input
id="time-quantum"
type="number"
min="1"
value={timeQuantum}
onChange={(e) => onTimeQuantumChange(parseInt(e.target.value) || 1)}
required
/>

)}
<Button
onClick={onRunSimulation}
disabled={disableRun}
className="w-full"
>
<Play className="mr-2 h-4 w-4" />
Run Simulation
</Button>

</CardContent>
</Card>
);
};
export default AlgorithmSelector;
Search
ENG
IN
16:45
07-05-2025

7.Output


clockwork-process-orchestrator/s
Process Scheduling Simulator
clockwork-process-orchestrator.vercel.app
All Bookmarks
Process Scheduling Simulator
Visualize CPU scheduling algorithms and analyze their performance metrics
Add Demo Processes
Clear All
Add Process
Process Name
P5
Arrival Time
0
Burst Time
1
Priority
2
Add Process
Scheduling Algorithm
Select Algorithm
First-Come, First-Served
Process List
Process
Color
Arrival Time
Burst Time
Priority
Actions
P1
0
2
4
P2
0
3
2
P3
0
4
2
P4
0
5
1
Ready to Simulate
Configure your algorithm settings and click "Run Simulation" to see the results.
Search
ENG
IN
16:48
07-05-2025


Process Scheduling Simulator
clockwork-process-orchestrator.vercel.app
Scheduling Algorithm
Select Algorithm
First-Come, First-Served
Run Simulation
Algorithm Guide
About First-Come, First-Served
Understanding Metrics
Simulation Results
Gantt Chart
Process Metrics
P1
P2
P3
P4
Average Waiting Time
4.00
Average Turnaround Time
7.50
CPU Utilization
100.00%
Simulation completed successfully
Process Scheduling Simulator - A tool to visualize CPU scheduling algorithms

Process Scheduling Simulator
clockwork-process-orchestrator.vercel.app
Add Process
P4
0
5
1
Scheduling Algorithm
Select Algorithm
First-Come, First-Served
Run Simulation
Algorithm Guide
About First-Come, First-Served
Understanding Metrics
Ready to Simulate
Configure your algorithm settings and click "Run Simulation" to see the results.
Process Scheduling Simulator - A tool to visualize CPU scheduling algorithms

8.Conclusion

The Process Scheduling Simulator provides a practical and interactive way to understand the core concepts of CPU scheduling in operating systems. By simulating different scheduling algorithms such as FCFS, SJN, Round Robin, and Priority Scheduling, the tool helps in visualizing how processes are handled by the CPU and how different strategies affect performance metrics like waiting time, turnaround time, and CPU utilization.

Through the simulation, users can experiment with various scheduling methods, gain insights into their advantages and limitations, and develop a deeper understanding of operating system behavior. The use of both tabular data and graphical representations (like the Gantt chart) makes it easier to analyze and compare the efficiency of different algorithms.

Overall, the simulator serves as an educational tool for students and developers, bridging the gap between theory and practice. It enhances learning by allowing users to experiment with real-world scheduling scenarios, fostering both theoretical knowledge and practical problem-solving skills.