**A logo with text overlay

AI-generated content may be incorrect.LOVELY PROFESIONAL UNIVERSITY, PUNJAB**

**CSE316-OPERATING SYSTEMS**

**Section:** K23DM

**Topic: Intelligent CPU Schedular Simulator**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No | Registration No | Name of Students | Roll No |
| 1. | 12318899 | K.Sai Vaishnav | 43 |
| 2. | 12319585 | Himanshu Magotra | 44 |
| 3. | 12307393 | Jayasri Karanam | 45 |

|  |  |
| --- | --- |
| Delivered By | Received By |
| K Sai Vaishnav Kumar (43) Himanshu Magotra (44)  Jayasri Lakshmi Karanam (45) | Dr. Gurbinder Singh |

1. **Project Overview**

The **Intelligent CPU Scheduler Simulator** is a software tool designed to help students and professionals understand the functionality of different CPU scheduling algorithms. Scheduling is a fundamental concept in operating systems that determines the order in which processes execute. This simulator provides a hands-on, interactive approach to learning by allowing users to input process details such as **arrival time, burst time, and priority**, and then simulating their execution using **First-Come-First-Serve (FCFS), Shortest Job First (SJF), Round Robin (RR), and Priority Scheduling** algorithms.

The simulator generates **real-time Gantt charts** to visually represent process execution sequences. Additionally, it calculates and displays key performance metrics, including **average waiting time, turnaround time, and response time**, enabling users to compare the efficiency of different scheduling techniques.

This project aims to bridge the gap between theoretical learning and practical implementation by providing a dynamic and user-friendly interface. It is developed using Java for backend computations, JavaFX or Swing for visualization, and optional web-based frameworks for an interactive experience. With **real-time visualization and analytical insights**, the simulator enhances the understanding of scheduling strategies, making it a valuable tool for educational and research purposes.

1. **Module-Wise Breakdown**

The **Intelligent CPU Scheduler Simulator** is divided into various modules to ensure an organized and efficient workflow. Each module performs a specific function to contribute to the overall execution of CPU scheduling algorithms. Below is a detailed breakdown of each module:

1. **User Input Module**
   * Allows users to enter process details including **arrival time, burst time, and priority**.
   * Provides an easy-to-use interface for input validation and error handling.
   * Supports **dynamic addition and removal** of processes before execution.
2. **Scheduling Algorithms Module**
   * Implements various CPU scheduling techniques: **FCFS, SJF, Round Robin, and Priority Scheduling**.
   * Contains logic for handling **pre-emptive and non-preemptive** scheduling.
   * Efficiently manages process execution order and context switching.
3. **Visualization Module**
   * Generates real-time **Gantt charts** to visually represent scheduling execution.
   * Displays process execution sequences using **color-coded bars**.
   * Provides an interactive UI for users to observe scheduling behavior.
4. **Performance Analysis Module**
   * Calculates key scheduling metrics such as **waiting time, turnaround time, and response time**.
   * Displays performance comparisons among different scheduling algorithms.
   * Helps users analyse and select the best scheduling approach based on system requirements.
5. **User Interface Module**
   * Provides a **graphical interface** for seamless interaction.
   * Uses **JavaFX or Swing** for building an interactive UI.
   * Ensures user-friendly navigation with minimal learning curve.
6. **Database/Storage Module (Optional)**
   * Stores user inputs and past scheduling results.
   * Allows users to reload previous computations for further analysis.
   * Utilizes simple file-based storage or a lightweight database like SQLite
7. **Performance Analysis Module**

* Calculates key scheduling metrics such as **waiting time, turnaround time, and response time**.
* Displays performance comparisons among different scheduling algorithms.
* Helps users analyse and select the best scheduling approach based on system requirements.

1. **User Interface Module**

* Provides a **graphical interface** for seamless interaction.
* Uses **JavaFX or Swing** for building an interactive UI.
* Ensures user-friendly navigation with minimal learning curve.

1. **Database/Storage Module (Optional)**

Stores user inputs and past scheduling results.

Allows users to reload previous computations for further analysis.

Utilizes simple file-based storage or a lightweight database like SQLite.

1. **Functionalities**

The **Intelligent CPU Scheduler Simulator** provides a range of functionalities to facilitate learning and understanding of CPU scheduling techniques. Below are the key functionalities of the simulator:

1. **Support for Multiple Scheduling Algorithms**
   * Implements **First-Come-First-Serve (FCFS), Shortest Job First (SJF), Round Robin (RR), and Priority Scheduling**.
   * Handles both **preemptive and non-preemptive** scheduling where applicable.
2. **User Input Handling**
   * Users can input **process details** such as arrival time, burst time, and priority.
   * Provides real-time **error validation** to ensure correct inputs.
   * Allows **modification and deletion** of process data before execution.
3. **Dynamic Visualization**
   * Generates **real-time Gantt charts** to represent scheduling execution.
   * Displays process execution using **color-coded blocks** for clarity.
   * Allows users to **pause, resume, or restart** the simulation.
4. **Performance Metrics Calculation**
   * Computes and displays **average waiting time, turnaround time, and response time**.
   * Provides a comparative analysis of different scheduling techniques.
   * Highlights **efficiency improvements** across scheduling strategies.
5. **User-Friendly Graphical Interface**
   * Designed using **JavaFX or Swing** for an intuitive experience.
   * Ensures **smooth navigation** between different features.
   * Supports **real-time data updates** as processes execute.
6. **Data Storage and Retrieval** 
   * Allows **saving and loading** of process data for further analysis.
   * Provides an option to **export Gantt charts** and scheduling results.
   * Uses **lightweight file storage or a small database** like SQLite for data persistence.
7. **Performance Comparison Feature**
   * Enables users to compare the efficiency of different algorithms.
   * Displays insights on which algorithm performs best under specific conditions.
   * Assists in selecting the optimal scheduling strategy for different workloads.
8. **GitHub Integration for Version Control**
   * Maintains version tracking of the project through **GitHub**.
   * Allows for easy collaboration and improvements.
   * Ensures structured and systematic code management.
9. **Technology Used**

The Intelligent CPU Scheduler Simulator is developed using a combination of programming languages, frameworks, and tools to ensure efficiency and usability. Below are the key technologies used:

**Programming Languages:**

* **Java** – Used for implementing scheduling logic and backend computations.
* **JavaScript** – Can be used for web-based visualizations if required.

**Libraries and Tools:**

* **JavaFX or Swing** – Used for developing the graphical user interface.
* **Matplotlib/D3.js** – Used for rendering Gantt charts and real-time visualizations.
* **JFreeChart** – Alternative for creating scheduling diagrams and performance graphs.

**Other Tools:**

* **GitHub** – Used for version control and collaborative development.
* **IntelliJ IDEA / VS Code** – Preferred IDEs for development.
* **SQLite or File Storage (Optional)** – For storing past simulation data.
* **JUnit** – Used for testing algorithm implementations to ensure correctness.

The combination of these technologies ensures that the simulator is **efficient, interactive, and easy to use**, making it a valuable tool for learning and experimentation in CPU scheduling.

1. A diagram of a software process

   AI-generated content may be incorrect.**FLOW CHART**
2. **Revision Tracking on GitHub**

To maintain version control and track development progress, we use **GitHub** as our repository management system. This ensures systematic organization, collaboration, and easy tracking of modifications throughout the project lifecycle.

**Repository Name:**

Intelligent-CPU-Scheduler-Simulator

**GitHub Link:**

**Himanshu:**

[**https://github.com/Himanshumagotra/Page-Replacement-**](https://github.com/Himanshumagotra/Page-Replacement-Simulator)[**Simulator**](https://github.com/Himanshumagotra/Page-Replacement-Simulator)

**Jayasrikaranam:**

[**https://github.com/Jayasrikaranam162**](https://github.com/Jayasrikaranam162)

**SaiVaishnavKumar:**

[**https://github.com/SaiVaishnavKumar**](https://github.com/SaiVaishnavKumar)

**Version Control Benefits:**

* **Commit History:** Tracks every change made to the code.
* **Branching & Merging:** Allows multiple features to be developed simultaneously.
* **Collaboration:** Enables teamwork and contribution tracking.
* **Issue Tracking:** Helps in identifying and fixing bugs efficiently.

1. **Conclusion**

* The **Intelligent CPU Scheduler Simulator** successfully provides an interactive and practical approach to learning CPU scheduling techniques. By allowing users to visualize process execution through **Gantt charts** and analyzing **performance metrics**, this simulator enhances the understanding of scheduling algorithms in a real-world scenario. The tool is designed to be user-friendly, efficient, and scalable, making it beneficial for students, educators, and professionals in the field of operating systems and computer science.
* This project helps users experiment with different scheduling algorithms and observe their behaviors under various scenarios. By comparing multiple techniques, users can gain insights into **which algorithm works best for specific workloads**, thus bridging the gap between theory and real-world application.
* In conclusion, the **Intelligent CPU Scheduler Simulator** serves as an essential learning aid for operating system concepts. With potential future improvements such as **multi-core scheduling, additional algorithms, machine learning integration, and a mobile version**, the project can evolve into a comprehensive scheduling analysis tool. This simulator not only reinforces theoretical knowledge but also prepares users for practical applications in computing environments.

1. **Future Scope**

The **Intelligent CPU Scheduler Simulator** has the potential for further development and enhancements. Some possible future improvements include:

1. **Multi-Core CPU Scheduling**
   * Implement scheduling techniques for **multi-core processors**.
   * Extend the simulator to handle **simultaneous execution of processes on multiple cores**.
2. **Addition of More Scheduling Algorithms**
   * Incorporate additional scheduling techniques such as **Multi-Level Queue Scheduling, Multi-Level Feedback Queue (MLFQ), and Lottery Scheduling**.
   * Provide **dynamic algorithm selection** based on workload requirements.
3. **Web-Based and Mobile Applications**
   * Develop a **web-based version** for accessibility across different platforms.
   * Create a **mobile application** to allow users to simulate CPU scheduling on their smartphones.
4. **Machine Learning-Based Scheduling Optimization**
   * Implement **AI/ML techniques** to suggest the most efficient scheduling algorithm based on input patterns.
   * Use predictive analytics to optimize scheduling strategies for real-world applications.
5. **Integration with Cloud Platforms**
   * Allow users to **run simulations on cloud-based platforms**.
   * Enable remote access and collaboration through **shared simulations**.
6. **References**
7. **Abraham Silberschatz, Peter B. Galvin, Greg Gagne**, "Operating System Concepts", Wiley.
8. **William Stallings**, "Operating Systems: Internals and Design Principles", Pearson.
9. **Andrew S. Tanenbaum**, "Modern Operating Systems", Pearson.
10. Research papers and online articles on CPU scheduling algorithms.
11. Java and JavaFX official documentation for UI development.
12. GitHub documentation for version control best practices.

**Appendix**

1. **AI-Generated Project Elaboration/Breakdown Report**

**1. Introduction**

The Intelligent CPU Scheduler Simulator is a software application designed to visualize and analyze various CPU scheduling algorithms used in operating systems. The simulator enables users to enter process details such as arrival time, burst time, and priority, then executes the selected scheduling algorithm while displaying a Gantt chart and performance metrics. This tool serves as an educational and analytical aid for students, researchers, and professionals in the field of operating systems and process management.

**2. Objectives**

* Implement an interactive CPU scheduling simulator for educational and analytical purposes.
* Provide real-time visualization of scheduling execution using a Gantt chart.
* Allow dynamic user input for process parameters (arrival time, burst time, priority).
* Implement the following CPU scheduling algorithms:
  + First-Come-First-Serve (FCFS)
  + Shortest Job First (SJF)
  + Round Robin (RR)
  + Priority Scheduling
* Compute and display performance metrics:
  + Average Waiting Time (AWT)
  + Turnaround Time (TAT)
  + Response Time
  + CPU Utilization
* Develop an easy-to-use Graphical User Interface (GUI) using JavaFX or Swing.

**3. Scope of the Project**

The project is targeted at:  
Students & Educators – To understand CPU scheduling techniques visually.  
Researchers – To analyze scheduling performance for different workloads.  
Developers – To explore optimization techniques in scheduling algorithms.

**4. Features and Functionalities**

1. **Process Input Handling**

* Users can add, edit, and remove processes dynamically.
* Ensures input validation to prevent errors.

1. **Algorithm Execution**

* Supports both preemptive and non-preemptive scheduling strategies.
* Executes the selected algorithm and calculates performance metrics.

1. **Gantt Chart Visualization**

* Displays an easy-to-read Gantt chart representing process execution over time.
* Uses color-coded bars for clear distinction.

1. **Performance Analysis**

* Computes and displays waiting time, turnaround time, and response time for each process.
* Provides comparative analysis for different scheduling algorithms.

1. **User-Friendly Interface**

* Intuitive UI designed using JavaFX or Swing for easy navigation.
* Interactive buttons for starting, pausing, and resetting simulations.

**5. Methodology**

Step 1: User enters process details (arrival time, burst time, priority).

Step 2: User selects a scheduling algorithm.

Step 3: The simulator processes the input and executes the algorithm.

Step 4: A Gantt chart is generated to visualize execution order.

Step 5: Performance metrics are calculated and displayed.

Step 6: Users can modify inputs and re-run the simulation.

**6. Expected Outcome**

* Users will gain hands-on experience with CPU scheduling algorithms.
* The simulator will provide visual clarity for scheduling execution.
* Users can compare different scheduling techniques using performance metrics.
* The project will act as an educational tool for operating system concepts.

**7. Challenges & Solutions**

* Challenge: Handling multiple scheduling algorithms efficiently.
* Solution: Implementing modular functions for each algorithm.
* Challenge: Designing an intuitive user interface.
* Solution: Using JavaFX/Swing for a user-friendly experience.
* Challenge: Ensuring accurate performance metric calculations.
* Solution: Applying validated formulas for time computations.

1. **Problem Statement**

**Background**

CPU scheduling is a fundamental concept in operating systems that determines the order in which processes are executed by the CPU. Efficient scheduling algorithms help optimize CPU utilization, reduce waiting time, and improve system performance. However, understanding and comparing various scheduling techniques can be challenging without proper visualization.

**Problem Definition**

The challenge is to develop an **Intelligent CPU Scheduler Simulator** that allows users to:

* Input multiple processes with **arrival time, burst time, and priority**.
* Simulate different **CPU scheduling algorithms** (FCFS, SJF, Round Robin, and Priority Scheduling).
* Visualize the execution sequence using a **Gantt chart**.
* Compute and display **performance metrics** like **waiting time, turnaround time, and response time**.

**Key Issues**

1. **Complexity in Understanding Scheduling** – Students and professionals find it difficult to grasp scheduling concepts theoretically.
2. **Lack of Visual Representation** – Without a graphical representation, understanding the flow of processes is challenging.
3. **Comparison of Algorithms** – Analyzing the efficiency of different scheduling techniques requires computational effort.

**Objective**

To design and develop an **interactive simulator** that effectively demonstrates **CPU scheduling techniques** with real-time visualization and performance evaluation.

1. **Code:**

**First Come First Serve (FCFS)**

class Process {

int pid, arrivalTime, burstTime, waitingTime, turnaroundTime;

}

public class FCFS {

public static void main(String[] args) {

Process[] processes = {

new Process(){ pid=1; arrivalTime=0; burstTime=4; },

new Process(){ pid=2; arrivalTime=1; burstTime=3; },

new Process(){ pid=3; arrivalTime=2; burstTime=1; }

};

int n = processes.length;

int currentTime = 0;

for (int i = 0; i < n; i++) {

Process p = processes[i];

if (currentTime < p.arrivalTime)

currentTime = p.arrivalTime;

p.waitingTime = currentTime - p.arrivalTime;

currentTime += p.burstTime;

p.turnaroundTime = p.waitingTime + p.burstTime;

}

for (Process p : processes) {

System.out.println("Process " + p.pid + ": Waiting Time = " + p.waitingTime + ", Turnaround Time = " + p.turnaroundTime);

}

}

}

**Shortest Job First (SJF-** **Non-Preemptive)**

import java.util.\*;

class Process {

int pid, arrivalTime, burstTime, waitingTime, turnaroundTime;

}

public class SJF {

public static void main(String[] args) {

List<Process> processes = new ArrayList<>();

processes.add(new Process(){ pid=1; arrivalTime=0; burstTime=6; });

processes.add(new Process(){ pid=2; arrivalTime=1; burstTime=2; });

processes.add(new Process(){ pid=3; arrivalTime=2; burstTime=8; });

int currentTime = 0;

List<Process> completed = new ArrayList<>();

while (!processes.isEmpty()) {

Process next = processes.stream()

.filter(p -> p.arrivalTime <= currentTime)

.min(Comparator.comparingInt(p -> p.burstTime))

.orElse(null);

if (next == null) {

currentTime++;

continue;

}

next.waitingTime = currentTime - next.arrivalTime;

currentTime += next.burstTime;

next.turnaroundTime = next.waitingTime + next.burstTime;

completed.add(next);

processes.remove(next);

}

for (Process p : completed) {

System.out.println("Process " + p.pid + ": Waiting Time = " + p.waitingTime + ", Turnaround Time = " + p.turnaroundTime);

}

}

}

**Round Robin (RR)**

import java.util.\*;

class Process {

int pid, arrivalTime, burstTime, remainingTime, waitingTime, turnaroundTime;

}

public class RoundRobin {

public static void main(String[] args) {

int quantum = 2;

Queue<Process> queue = new LinkedList<>();

List<Process> processes = Arrays.asList(

new Process(){ pid=1; arrivalTime=0; burstTime=5; remainingTime=5; },

new Process(){ pid=2; arrivalTime=1; burstTime=4; remainingTime=4; },

new Process(){ pid=3; arrivalTime=2; burstTime=2; remainingTime=2; }

);

int time = 0;

queue.addAll(processes);

while (!queue.isEmpty()) {

Process p = queue.poll();

if (p.arrivalTime > time) {

time = p.arrivalTime;

}

int execTime = Math.min(quantum, p.remainingTime);

p.remainingTime -= execTime;

time += execTime;

if (p.remainingTime > 0) {

p.arrivalTime = time;

queue.offer(p);

} else {

p.turnaroundTime = time - p.arrivalTime + (p.burstTime - p.remainingTime);

p.waitingTime = p.turnaroundTime - p.burstTime;

}

}

for (Process p : processes) {

System.out.println("Process " + p.pid + ": Waiting Time = " + p.waitingTime + ", Turnaround Time = " + p.turnaroundTime);

}

}

}

**Priority Scheduling(Non-preemptive)**

import java.util.\*;

class Process {

int pid, arrivalTime, burstTime, priority, waitingTime, turnaroundTime;

}

public class PriorityScheduling {

public static void main(String[] args) {

List<Process> processes = new ArrayList<>();

processes.add(new Process(){ pid=1; arrivalTime=0; burstTime=5; priority=2; });

processes.add(new Process(){ pid=2; arrivalTime=1; burstTime=3; priority=1; });

processes.add(new Process(){ pid=3; arrivalTime=2; burstTime=4; priority=3; });

processes.sort(Comparator.comparingInt(p -> p.priority)); // lower value = higher priority

int currentTime = 0;

for (Process p : processes) {

if (currentTime < p.arrivalTime)

currentTime = p.arrivalTime;

p.waitingTime = currentTime - p.arrivalTime;

currentTime += p.burstTime;

p.turnaroundTime = p.waitingTime + p.burstTime;

}

for (Process p : processes) {

System.out.println("Process " + p.pid + ": Waiting Time = " + p.waitingTime + ", Turnaround Time = " + p.turnaroundTime);

}

}

}