Jaypee Institute of Information Technology, Noida
Project Synopsis

Project Title:- AI Enhanced CPU Scheduler

Operating System and System Programming Lab
15B17CI472

Submitted By:

Ayansh Jain	23103214
Ayush Agarwal	23103220
Poorvi Tandon	23103234

Submitted To:
Dr. Anil Kumar Mehto

ABSTRACT

The **AI-Enhanced CPU Scheduler** is an intelligent process scheduling framework that integrates **machine learning techniques** with traditional CPU scheduling algorithms to optimize task execution in dynamic computing environments. Unlike static schedulers such as FCFS, SJF, or Priority Scheduling, this system continuously learns from historical workload patterns, process characteristics, and system states to make **predictive, adaptive scheduling decisions**. By leveraging models like **reinforcement learning** or **neural networks**, the scheduler can anticipate process burst times, adjust priorities in real time, and balance trade-offs between throughput, latency, and fairness. This approach is particularly beneficial in **heterogeneous and real-time systems**, where workload variability and strict deadlines demand more responsive scheduling strategies. The project aims to demonstrate measurable improvements in **CPU utilization, turnaround time, and energy efficiency** compared to conventional methods.

Scope of the Project:-

A. Functional Requirements

1. **Process Input & Management** — Accept process details (arrival, burst time, priority) via an intuitive interface; maintain ready queues and update process states.
2. **Baseline & AI-Driven Scheduling** — Implement standard algorithms (FCFS, SJF, RR, Priority) alongside an AI-based decision engine that predicts and adapts scheduling in real time.
3. **Dynamic Priority Adjustment** — Modify process priorities during execution based on workload patterns and system state.
4. **Performance Analysis & Visualization** — Compute metrics (waiting time, turnaround time, CPU utilization, throughput) and visualize results with Gantt charts and dashboards.

B. Non-Functional Requirements

1. **Performance & Scalability** — Efficiently handle large workloads (e.g., up to 10,000 processes) without major slowdown.
2. **Usability** — Provide a clean, interactive interface requiring minimal user training.
3. **Portability** — Ensure cross-platform compatibility across Windows, Linux, and macOS.

C. Modules

1. **User Interface Module** — Interactive input forms and result displays, supporting manual entry and file import.
2. **Process Management Module** — Stores process details, manages queues, updates states.
3. **Scheduler Core Module** — Houses both traditional and AI-based scheduling logic for decision-making.
4. **Metrics & Analytics Module** — Calculates and stores performance results for comparison.
5. **Visualization Module** — Generates Gantt charts, comparison graphs, and summary tables.

Tools and Technologies Used

- **Programming Language:** Python (or Java/C++) for implementing scheduling algorithms, AI models, and core logic.
- **Machine Learning Frameworks:** Scikit-learn, TensorFlow, or PyTorch for building and training predictive models to enhance scheduling decisions.
- **Frontend Technologies (if web-based):** HTML5, CSS3, and JavaScript (with React.js or Vue.js) for creating the interactive user interface.
- **Visualization Libraries:** Matplotlib, Chart.js, or D3.js to create Gantt charts, performance graphs, and dashboards.
- **Database:** SQLite or PostgreSQL for storing process logs, datasets, and AI training data.
- **Version Control:** Git with GitHub or GitLab for source code management and team collaboration.
- **Testing Tools:** PyTest (Python) or JUnit (Java) for unit, integration, and performance testing.
- **IDE/Editor:** Visual Studio Code, IntelliJ IDEA, or PyCharm for development.
- **Operating System Support:** Cross-platform functionality on Windows,Linux and macOS,
- **Deployment/Testing Environments :** Docker or VirtualBox to run the project in isolated, consistent setups.

Design of the Project :-

The project follows a **modular layered architecture** for separation of concerns:

1. **Presentation Layer**
    - User interface for input/output
    - Interactive forms for entering process details
    - Live charts for visual feedback
2. **Application Layer**
    - Scheduler engine (Baseline + AI module)
    - Process management and queue handling
3. **Data Layer**
    - Dataset storage (process logs, AI training history)
    - Model persistence (saved ML models for reuse)
4. **Visualization Layer**
    - Gantt chart rendering
    - Comparative performance graph generation

<u>Implementation Details :</u>

Core Components Implemented:

1. **Process Data Structures** — Custom classes/structs to store attributes (PID, arrival time, burst time, priority, deadlines).
2. **Baseline Algorithms** — FCFS, SJF, SRTF, RR, Priority Scheduling implemented as separate modules for clarity and testing.
3. **AI Module** — Predicts optimal job order using ML regression/classification models trained on synthetic or real datasets.
4. **Dynamic Priority Handler** — Adjusts task priorities based on workload context and performance goals.
5. **Metrics Calculator** — Computes waiting time, turnaround time, CPU utilization, throughput, and (if simulated) power usage.
6. **Visualization Engine** — Generates Gantt charts and metric comparison graphs on demand.
7. **File Handling** — Import/export process lists (CSV/JSON) and store logs for AI training.
8. **Integration Layer** — Combines AI predictions with scheduler logic, falling back to baseline in case of confidence drop.

REFERENCES:

Operating System Concepts (Silberschatz et al.)

Bootstrap Documentation:

Sidhu, Armaan. (2023). Process Scheduling in Operating Systems and Evolution of Windows (2023). 10.fi0S4/m9.figshare.23537031.v1.

Ramamritham, Krithivasan & Stankovic, John. (1994). Scheduling Algorithms and Operating Systems Support for Real-Time System. Proceedings of the IEEE. 52. 55 -67. 10.1109/5.259426.