A SYNOPSIS ON

PROCESS SCHEDULER

Submitted in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

In

Computer Science & Engineering

Submitted by:

Priya Koranga	2261433
Kareena Aithani	2261313
Chetna Karayat	2261157
Saniya Chaniyal	2261511

Under the Guidance of

Mr. Prince Kumar

Assistant Professor

Project Team ID: 97

Department of Computer Science & Engineering

Graphic Era Hill University, Bhimtal, Uttarakhand

March-2025

Graphic Era Hill University
BHIMTAL CAMPUS

CANDIDATE'S DECLARATION

We hereby certify that the work which is being presented in the Synopsis entitled **"Process Scheduler"** in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science & Engineering of the Graphic Era Hill University, Bhimtal campus and shall be carried out by the undersigned under the supervision of **Mr. Prince Kumar, Assistant Professor**, Department of Computer Science & Engineering, Graphic Era Hill University, Bhimtal.

Priya Koranga	2261433	signature
Kareena Aithani	2261313	signature
Chetna Karayat	2261157	signature
Saniya Chaniyal	2261511	signature

The above mentioned students shall be working under the supervision of the undersigned on the **"Process Scheduler"**

Signature	Signature
Supervisor	**Head of the Department**

Internal Evaluation (By DPRC Committee)

Status of the Synopsis: Accepted / Rejected

Any Comments:

Name of the Committee Members: **Signature with Date**

1.

2.

Table of Contents

Chapter 1

Introduction and Problem Statement

In the following sections, a brief introduction and the problem statement for the work has been included.

1. Introduction

In modern computing environments, efficient process scheduling is essential to ensure optimal utilization of CPU resources, fair execution of processes, and improved system responsiveness. As multitasking operating systems execute multiple processes simultaneously, a scheduling mechanism is required to **determine the order in which processes are allocated CPU time**. Effective scheduling ensures that critical tasks receive priority, system throughput remains high, and user experience is seamless.

Traditional process scheduling algorithms such as **First-Come-First-Serve (FCFS), Shortest Job First (SJF), Round Robin (RR), and Priority Scheduling** provide fundamental process management but have notable limitations in handling complex, dynamic, and real-time workloads. These conventional techniques often fail to adapt to **changing system conditions**, leading to **inefficiencies, increased waiting times, process starvation, and poor CPU utilization**.

The **advanced Process Scheduler** project aims to develop a **dynamic, intelligent, and adaptive process scheduling system** that overcomes the shortcomings of traditional scheduling techniques. This project will integrate **modern scheduling algorithms**, **real-time execution handling**, and **machine learning-based optimizations** to enhance scheduling performance. Key features of this project include:

- ❖ **Multilevel Feedback Queue (MLFQ):** A dynamic scheduling mechanism that adjusts process priority based on execution history.
- ❖ **Completely Fair Scheduler (CFS):** A fairness-based scheduling approach ensuring equal CPU time distribution among processes.
- ❖ **Earliest Deadline First (EDF):** A real-time scheduling algorithm that prioritizes tasks based on deadlines.
- ❖ **AI-Driven Scheduling Optimization:** Leveraging machine learning techniques to predict and select the most efficient scheduling algorithm based on historical workload data.
- ❖ **Graphical User Interface (GUI):** An interactive visualization tool that allows users to analyze and compare scheduling outcomes in real-time.

Additionally, this scheduler will provide support for **preemptive and non-preemptive scheduling**, **multiprocessing/multithreading**, and **dynamic workload adjustments**. The ultimate goal of this project is to create an **intelligent and interactive scheduling system** that can adapt to different scenarios, improve system efficiency, and enhance user control over process execution.

2. Problem Statement

The problem statement for the present work can be stated as follows:

In real-world computing systems, multiple processes compete for CPU resources, making efficient scheduling essential to maintaining performance and responsiveness. Traditional scheduling algorithms operate using predefined rules that often do not adapt well to dynamic workloads. As a result, issues such as **high turnaround time, process starvation, inefficient CPU usage, and unfair execution priority** arise.

Additionally, conventional schedulers lack **real-time visualization and interactive user control**, making it difficult for users to monitor and analyze scheduling decisions. Furthermore, **modern computing demands** require schedulers to optimize for **multiprocessing, real-time execution, and AI-based improvements**, which are not adequately addressed by traditional scheduling techniques.

Challenges in Traditional Scheduling Systems

Several critical challenges in existing process scheduling techniques highlight the need for an advanced scheduling solution:

- ❖ **Inefficient CPU Utilization** – Many scheduling techniques lead to CPU idle time or inefficient execution order, reducing overall system performance.
- ❖ **High Waiting and Turnaround Time** – Inefficient scheduling may cause longer delays for certain processes, affecting response time and throughput.
- ❖ **Process Starvation & Priority Inversion** – Low-priority processes may experience indefinite delays, and high-priority tasks might not execute on time.
- ❖ **No Dynamic Adaptation** – Most scheduling algorithms work on predefined rules and do not adapt dynamically to workload changes.
- ❖ **Lack of Real-time Visualization & User Control** – Existing schedulers do not allow users to interact with the scheduling process or visualize real-time execution.
- ❖ **Limited Support for Parallel Processing & Multiprocessing** – Many schedulers do not effectively utilize multiple CPU cores.
- ❖ **AI-based Optimization** – Existing schedulers do not leverage machine learning to predict and optimize scheduling decisions dynamically.

To overcome these limitations, the advanced Process Scheduler project will:

- ❖ Implement **adaptive, preemptive, and real-time scheduling algorithms** to optimize CPU performance.
- ❖ Provide a **graphical user interface (GUI) for real-time visualization** of scheduling decisions.
- ❖ Support **multiprocessing/multithreading** to handle concurrent execution efficiently.
- ❖ Utilize **machine learning techniques** to predict and optimize scheduling dynamically based on historical workload patterns.
- ❖ Allow **user interaction**, enabling manual modification of process priorities, execution order, and scheduling policies.
- ❖ Analyze key **performance metrics** such as turnaround time, waiting time, and CPU utilization to compare different scheduling techniques.

  The **advanced Process Scheduler** will be designed as a **simulation tool** that can be used for educational purposes, research, and real-world scheduling optimizations. The project will:

  - ❖ Help users **understand and compare scheduling algorithms** in real-time.
  - ❖ Provide an **interactive platform** for experimenting with different scheduling policies.
  - ❖ Enable **AI-driven scheduling enhancements** for better system performance.
  - ❖ Improve **process execution efficiency** by implementing **intelligent scheduling techniques**.

By introducing **dynamic, AI-driven, and interactive scheduling**, this project aims to create an **efficient, adaptive, and user-friendly process scheduler** for modern computing systems.

Process scheduling is a crucial aspect of operating system design, directly influencing system responsiveness, performance, and CPU efficiency. However, traditional scheduling techniques **lack adaptability, AI-driven optimization, and real-time interactivity**, making them inefficient for modern computing needs.

The **Advanced Process Scheduler** will bridge this gap by implementing **intelligent, interactive, and AI-powered scheduling strategies**, providing a **highly optimized, user-friendly, and efficient scheduling solution**. This project will serve as a **practical and research-oriented tool**, demonstrating how **modern scheduling techniques can significantly enhance process execution in complex, multitasking environments**.

Chapter 2

Background/ Literature Survey

Background

Process scheduling is a crucial function of modern **operating systems (OS)**, responsible for managing the execution of multiple processes efficiently. In a multitasking environment, multiple programs and tasks compete for CPU resources, making it essential for the OS to determine the optimal execution order. Effective scheduling ensures **optimal CPU utilization, minimal waiting time, and balanced resource distribution** among processes.

Traditional process schedulers follow predefined algorithms such as **First-Come-First-Serve (FCFS), Shortest Job First (SJF), Round Robin (RR), and Priority Scheduling**. While these techniques work well under specific conditions, they **lack adaptability to modern real-time computing environments**. Today's computing demands involve **parallel processing, real-time scheduling, and dynamic workload balancing**, requiring **advanced scheduling mechanisms** that are **flexible, intelligent, and adaptive**.

The **advanced Process Scheduler** is designed to overcome the limitations of traditional schedulers by integrating:

- ❖ **Adaptive Scheduling Mechanisms** – Adjusts dynamically to workload changes.
- ❖ **Artificial Intelligence (AI) & Machine Learning (ML)** – Predicts process execution patterns and optimizes scheduling.
- ❖ **Multi-core & Parallel Processing Support** – Enhances CPU efficiency in modern computing systems.
- ❖ **Real-time Monitoring & User Control** – Allows interactive scheduling with visualization tools.

This project aims to develop a **high-performance, intelligent, and user-friendly process scheduler** that optimizes **CPU usage, minimizes latency, and ensures fairness** among processes.

Evolution of Process Scheduling:

Early Scheduling Alogrithms:

The earliest scheduling algorithms were simple and static, designed for batch processing environments. These include:

- ❖ **First-Come-First-Serve (FCFS)** – The first process to arrive is executed first, but it suffers from the **convoy effect**, where shorter tasks get delayed by longer ones.
- ❖ **Shortest Job First (SJF)** – Executes processes with the smallest burst time first, reducing average waiting time, but **requires prior knowledge of execution time**.
- ❖ **Round Robin (RR)** – Assigns each process a fixed **time quantum**, ensuring fairness but potentially causing **excessive context switching**.
- ❖ **Priority Scheduling** – Executes processes based on assigned priority levels, leading to **starvation of lower-priority processes** unless handled properly.

The Need for Advanced Scheduling

While these traditional methods are still widely used, they **fail to address modern computing challenges**, including:

- ❖ **High system load:** Static scheduling policies cannot efficiently distribute CPU time when process loads fluctuate.
- ❖ **Parallel execution:** Single-CPU schedulers struggle to efficiently allocate tasks in **multi-core architectures**.
- ❖ **Real-time execution:** Many schedulers do not guarantee deadlines for **real-time systems**.
- ❖ **Inefficient resource utilization:** Static schedulers may lead to **underutilized CPU cycles** or excessive **context switching overhead**.

To address these challenges, modern schedulers integrate **adaptive, AI-driven decision-making** that dynamically **adjusts execution policies based on real-time system conditions**. The evolution of process scheduling from **basic static algorithms to intelligent, adaptive systems** has significantly improved **CPU efficiency, fairness, and response times**. However, traditional schedulers **struggle to adapt to dynamic workloads, real-time execution, and multi-core processing**.The **Advanced Process Scheduler** will integrate **modern AI-based scheduling techniques, real-time monitoring, and user-driven customization** to provide a **highly efficient and flexible scheduling system**. This approach will:
Improve **CPU utilization** in modern **multi-core environments**.
Enhance **fairness and efficiency** by preventing starvation and bottlenecks.
Provide **real-time execution control** through **interactive visualization tools**.

By leveraging **machine learning, real-time analytics, and advanced scheduling algorithms**, this project aims to **redefine process scheduling efficiency in modern computing environments**.

Literature Survey

Process scheduling has been a fundamental topic in operating system research, evolving from traditional static algorithms to modern **dynamic and AI-driven scheduling techniques**. This literature survey explores key advancements in process scheduling, highlighting their strengths and limitations.

1.Traditional Scheduling Algorithms

Early research in process scheduling focused on fundamental scheduling policies such as:

- **First-Come-First-Serve (FCFS):** Simple but suffers from the **convoy effect**, leading to inefficient CPU usage.
- **Shortest Job First (SJF):** Minimizes waiting time but **requires prior knowledge of job execution time**.
- **Round Robin (RR):** Ensures fairness but **performance depends on the chosen time quantum**.
- **Priority Scheduling:** Assigns priorities to processes but **risks starvation of lower-priority tasks**.

These traditional techniques work well in simple environments but **lack adaptability** for real-time and multi-core processing.

2. Dynamic & Adaptive Scheduling Techniques

To overcome the limitations of static scheduling, researchers developed **dynamic scheduling techniques** such as:

- **Multilevel Feedback Queue (MLFQ):** Adjusts process priority dynamically based on execution history, improving fairness and responsiveness.
- **Completely Fair Scheduler (CFS):** Used in Linux, it **ensures fair CPU allocation** by using a red-black tree to track process execution time.
- **Real-Time Scheduling (RTS):** Algorithms like **Earliest Deadline First (EDF)** and **Rate Monotonic Scheduling (RMS)** prioritize real-time task execution but require precise time constraints.

3.AI & Machine Learning in Scheduling

Recent studies have explored **AI-driven scheduling techniques**, leveraging machine learning to optimize CPU allocation:

- **Reinforcement Learning-Based Scheduling:** Dynamically adjusts scheduling parameters based on system conditions.
- **Neural Networks for Scheduling Prediction:** Predicts **optimal time slices** for Round Robin scheduling.
- **Genetic Algorithms for Scheduling Optimization:** Uses evolutionary principles to minimize **CPU idle time** and enhance fairness.

Chapter 3

Objectives

The objectives of the proposed work are as follows:

Process scheduling is a fundamental component of operating systems, ensuring efficient CPU resource allocation among multiple processes. The primary goal of a **process scheduler** is to maximize CPU utilization, minimize waiting time, and provide fair execution for all processes.

Traditional scheduling algorithms like **First-Come-First-Serve (FCFS), Shortest Job First (SJF), Round Robin (RR), and Priority Scheduling** have been widely used but exhibit limitations in modern computing environments. These challenges include **high system load, parallel execution inefficiencies, real-time constraints, and the need for AI-driven decision-making**.

The **Advanced Process Scheduler** aims to address these challenges by integrating **AI-based predictive scheduling, real-time monitoring, and adaptive scheduling mechanisms**. The following objectives highlight the core aspects of this project.

Key Objectives

1. Enhancing CPU Utilization and Efficiency

- Develop an **optimized scheduling system** that minimizes CPU idle time and improves overall efficiency.
- Reduce **waiting time and turnaround time** by dynamically selecting the best scheduling algorithm
- Implement **parallel processing** to enhance performance in **multi-core** and **multi-threaded** environments.
- Optimize scheduling strategies to **minimize context-switching overhead** and improve response time.

2. Implementation of Adaptive and AI-Driven Scheduling

- Design a **self-learning scheduler** that adapts scheduling policies based on workload patterns using **machine learning (ML) techniques**.
- Use **predictive analytics** to anticipate process execution time and dynamically adjust priorities.
- Incorporate **neural networks** and **reinforcement learning** to optimize scheduling policies based on real-time execution feedback.

3. Support for Multi-Core and Parallel Processing

- Develop an **intelligent load balancing mechanism** that evenly distributes processes across multiple CPU cores.

- Implement **symmetric and asymmetric multi-processing (SMP & AMP) scheduling strategies** for efficient process execution.
- ⍰ Improve parallelism by **allocating resources dynamically** based on process dependencies and execution patterns.

4. Real-Time Process Scheduling for Critical Applications

- Incorporate **real-time scheduling algorithms** such as:
  Earliest Deadline First (EDF) – Ensures priority execution of time-critical tasks.
- **Rate Monotonic Scheduling (RMS)** – Guarantees periodic task execution.
  Deadline-Based Scheduling – Prevents deadline violations in embedded and industrial applications.
- Implement **dynamic priority adjustments** to ensure real-time responsiveness in **mission-critical systems**.

5. Fairness, Starvation Prevention, and Priority Management

- Ensure **fair CPU allocation** by implementing an **adaptive priority scheduler**.
- Prevent starvation by gradually **increasing the priority of long-waiting processes**.
- Develop a **hybrid scheduling policy** that balances **fairness and efficiency** based on system workload.

6. Development of an Interactive Scheduling Dashboard

- Provide a **Graphical User Interface (GUI)** for real-time monitoring of process execution.
- Display **CPU usage statistics, Gantt charts, waiting time analysis, and turnaround time** in a user-friendly format.
- Allow users to manually adjust scheduling policies and compare their performance in real-time.

Innovative Features and Enhancements

1. AI-Based Scheduling Optimization

- Implement **reinforcement learning (RL)** to enhance scheduling efficiency based on historical data.
- Use **genetic algorithms (GA)** to optimize CPU allocation and scheduling performance.
- Apply **deep learning models** to predict **optimal time quantum selection** for Round Robin scheduling.

2. Energy-Efficient Process Scheduling

- Implement **dynamic frequency scaling (DFS)** to adjust CPU speed based on process load.

- Reduce power consumption by scheduling low-priority tasks during **low-energy consumption cycles**.
- Improve **battery efficiency in mobile and embedded systems** by optimizing resource allocation.

3. Fault Tolerance and Robust Error Handling

- Integrate **failover mechanisms** to prevent system crashes due to scheduling failures.
- Implement **self-recovery features** that detect and correct scheduling errors dynamically.
- Develop **error handling policies** to minimize disruptions in case of **CPU overloads or process failures**.

4. Cloud-Based and Distributed Scheduling Support

- Extend scheduling mechanisms to **cloud computing environments** to efficiently manage virtual machines (VMs).
- Implement **load balancing for distributed systems** to optimize execution across multiple data centers.
- Enable **dynamic resource allocation** to handle variable workloads in cloud environments.

Expected Outcomes

The **advanced Process Scheduler** will provide:

- **Higher CPU Utilization** – Maximizes CPU performance with **intelligent scheduling techniques**.
- **Reduced Latency and Context Switching** – Optimizes process execution order to minimize overhead.
- **Fair and Efficient Scheduling** – Prevents starvation while ensuring optimal execution for all processes.
- **Improved Real-Time Performance** – Supports critical applications with **deadline-aware scheduling**.
- **Scalability and Cloud Support** – Enables efficient scheduling in **multi-core, distributed, and cloud-based environments**.
- **Energy-Efficient Execution** – Reduces power consumption through optimized CPU resource allocation.
- **Interactive and User-Friendly Experience** – Provides real-time visualization and manual control options for users.

The **advanced Process Scheduler** aims to **redefine process scheduling by integrating AI-driven, real-time, and adaptive scheduling strategies**. The project will introduce:

- **Intelligent Scheduling Mechanisms** that optimize CPU utilization and process efficiency.
- **Dynamic Priority Adjustment** to improve fairness and prevent starvation.
- **Real-Time Monitoring & User Interaction** for enhanced process control.
- **Scalability & Multi-Core Support** for modern computing architectures.
- **Energy-Efficient & Fault-Tolerant Design** to improve reliability and sustainability.

By addressing the challenges of traditional scheduling methods and **leveraging AI and real-time analytics**, this project will contribute significantly to **operating system research, cloud computing, embedded systems, and high-performance computing environments**.

The **Advanced Process Scheduler** is a transformative project that goes beyond **traditional CPU scheduling techniques**. With its integration of **AI-driven adaptability, real-time execution monitoring, and intelligent resource management**, this scheduler is designed to **meet the demands of modern computing environments**.

Key innovations include:

- **AI-Powered Scheduling Algorithms** – Reducing latency and improving efficiency.
- **Real-Time Analytics & Monitoring** – Allowing user-driven optimizations.
- **Fair & Adaptive Scheduling** – Preventing starvation and ensuring equitable resource distribution.
- **Multi-Core & Distributed Processing Support** – Optimizing execution in cloud and high-performance computing systems
- . **Energy-Efficient & Fault-Tolerant Execution** – Enhancing sustainability and reliability.

As we continue to explore **AI, cloud computing, blockchain, and quantum technologies**, the **future of process scheduling** will evolve to **become more intelligent, efficient, and adaptable to complex computing needs**.

This project serves as **a foundational step toward the next generation of scheduling techniques**, aiming to significantly improve **performance, reliability, and user control** in modern operating systems.

Chapter 4

Hardware and Software Requirements

1. Hardware Requirements

Sl. No	Name of the Hardware	Specification
1.	Processor (CPU)	Intel Core i5, Intel Core i7
2.	RAM	8 GB, 16 GB DDR4 or higher
3.	Storage	256 GB SSD, 512 GB SSD
4.	Graphics (GPU)	Integrated Graphics,
5.	Operating System	Windows 10/11, Linux (Ubuntu 20.04+), ma OS.

2 .Software Requirements

Sl. No	Name of the Software	Specification
1.	Operating System	Windows / Linux / macOS
2.	Programming Language, Web Interface, AI/ML Libraries	Python / Java / C, HTML, CSS, Flask, React.js, Django, FastAPI, TensorFlow / PyTorch, Scikit-Learn
3.	Development Environment, Database, Simulation & Visualization, Parallel Processing Support	IDE / Code Editor, VS Code, PyCharm, MySQL / PostgreSQL / SQLite, Gantt Chart / Graph Plotting, Matplotlib, Tkinter, Python threading, Java ExecutorService.

Chapter 5

Possible Approach/ Algorithms

Process scheduling is a crucial component of modern operating systems that determines the execution order of processes. The **advanced Process Scheduler** aims to enhance traditional scheduling techniques by integrating **AI-driven optimizations, multi-core processing, real-time adaptability, and cloud-based task management**.

This section explores **various scheduling approaches**, from **classical algorithms** to **AI-powered models**, and discusses their suitability for different computing environments.

1. **Classical Scheduling Approaches**

Traditional CPU scheduling algorithms are widely used in operating systems and can serve as a foundation for more advanced techniques.

1.1. First-Come, First-Served (FCFS)

- **Approach:** Processes are executed in the order they arrive (FIFO queue).
- **Advantages:** Simple and fair.
- **Disadvantages: High waiting time**, leading to the **convoy effect** (slow processes delaying faster ones).

1.2. Shortest Job Next (SJN) / Shortest Job First (SJF)

- **Approach:** Executes the process with the shortest CPU burst time first.
- **Advantages: Minimizes average waiting time.**
- **Disadvantages:** Requires knowing burst times in advance, leading to **starvation** of longer processes.

1.3. Round Robin (RR)

- **Approach:** Each process gets a **fixed time slice (quantum)** in a cyclic order.
- **Advantages: Ensures fairness**, prevents starvation.
- **Disadvantages:** Poor performance with **improper time quantum** selection.

1.4. Priority Scheduling

- **Approach:** Processes are assigned a priority, and higher-priority processes execute first.
- **Advantages:** Allows important processes to execute first.
- **Disadvantages: Starvation** of low-priority tasks (mitigated using **aging**).

1.5. Multi-Level Queue (MLQ) and Multi-Level Feedback Queue (MLFQ)

- **Approach:** Divides processes into multiple priority queues and allows **dynamic queue adjustments** in MLFQ.
- **Advantages: Combines multiple scheduling techniques**, adaptive.
- **Disadvantages:** Complex to implement.

2. Advanced Scheduling Approaches

2.1. AI-Based Predictive Scheduling (Machine Learning Approach)

- **Approach:** Uses **historical data and real-time process metrics** to predict and optimize scheduling decisions.
- **Algorithm:**
    1. Collect **CPU burst history** and process attributes.
    2. Use **Supervised Learning (Neural Networks, Decision Trees, Random Forest)** to predict burst times.
    3. Dynamically adjust scheduling policies based on predictions.
- **Advantages: Adapts to workload patterns**, **reduces average waiting time**.
- **Disadvantages:** Requires **training data**, **computational overhead**.

2.2. Parallel and Multi-Core Scheduling

- **Approach:** Uses multi-threaded execution to distribute processes across multiple CPU cores.
- **Algorithm:**
    1. Implement **Load Balancing Algorithms** (e.g., Work Stealing, Round-Robin Load Balancing).
    2. Assign tasks to CPU cores dynamically based on **current CPU utilization**.
- **Advantages:** Increases **CPU efficiency**, **better resource utilization**.
- **Disadvantages:** Complex **synchronization mechanisms** required.

2.3. Real-Time Scheduling (Rate-Monotonic & Earliest Deadline First)

- **Approach:** Used for **real-time systems**, where tasks must complete within strict deadlines.
- **Algorithms:**
    - **Rate-Monotonic Scheduling (RMS):** Assigns **static priorities** based on task frequency.
    - **Earliest Deadline First (EDF):** Dynamically schedules the process with the closest deadline.
- **Advantages:** Suitable for **embedded and IoT systems**.
- **Disadvantages:** Can **fail under high CPU loads**.

Comparative Analysis of Approaches

Approach	**Best for**	**Advantages**	**Disadvantages**
FCFS	Simple, batch processing	Easy to implement	High waiting time
SJF	Short jobs, low latency	Optimized for minimal waiting	Starvation of long jobs
Round Robin	Time-sharing OS	Fair process execution	Inefficient for large workloads
MLFQ	Interactive & real-time systems	Adaptive, efficient	Complex implementation
AI-Based Predictive Scheduling	Cloud, dynamic environments	Self-optimizing, real-time adjustments	High computational cost
Parallel Scheduling	Multi-core & distributed systems	Maximizes CPU usage	Requires thread synchronization

Table 1: Pseudo code of the FCFS algorithm

```
Algorithm FCFS_Scheduler(processes[], n):
1. Sort processes by arrival_time.
2. Initialize start_time = processes[0].arrival_time.
3. For each process:
   a. Compute wait_time = start_time - process.arrival_time.
   b. Execute process.
   c. Update start_time += process.burst_time.
4. Compute and print average waiting time.
```

Table 2: Pseudo code of the SJF algorithm

```
Algorithm SJF_Scheduler(processes[], n):
1. Sort processes by burst_time.
2. Initialize start_time = 0.
3. For each process:
    a. Compute wait_time = start_time - process.arrival_time.
    b. Execute process.
    c. Update start_time += process.burst_time.
4. Compute and print average waiting time.
```

Table 3: Pseudo code of the RR algorithm

```
Algorithm Round_Robin(processes[], n, time_quantum):
1. Initialize queue with processes[].
2. While queue is not empty:
    a. Dequeue process.
    b. If process.burst_time > time_quantum:
        i. Execute for time_quantum.
        ii. Reduce burst_time.
        iii. Enqueue process back.
    c. Else:
        i. Execute completely.
3. Compute and print execution details.
```

Table 4: Pseudo code of the MLFQ algorithm

```
Algorithm MLFQ_Scheduler(processes[], queues[], quantum[]):
1. Assign processes to the highest priority queue.
2. While any queue is not empty:
    a. For each queue level from high to low:
        i. While queue is not empty:
            - Dequeue process.
            - If burst_time > quantum[level]:
                - Execute for quantum[level].
                - Reduce burst_time.
                - Move process to lower priority queue.
            - Else, execute completely.
3. Print execution summary.
```

Table 5: Pseudo code of the AI-Based Predictive Scheduling algorithm

```
Algorithm Train_AI_Scheduler(training_data):
1. Train a Machine Learning model using training_data.
2. Return trained model.


Algorithm AI_Scheduler(processes[], n, model):
1. For each process:
   a. Predict burst_time using model.
2. Sort processes by predicted_burst_time.
3. Execute processes sequentially.
```

Table 6: Pseudo code of the Parallel Scheduling for Multi-Core Processors algorithm

```
Algorithm Parallel_Scheduler(processes[], num_cores):
1. Assign each process to the least loaded core.
2. While processes remain:
   a. Parallel Execution:
      i. For each core:
         - Execute assigned process.
   b. Assign new processes dynamically to available cores.
```

Table 7: Pseudo code of the Real-Time Scheduling (Earliest Deadline First – EDF) algorithm

```
Algorithm EDF_Scheduler(processes[], n):
1. Sort processes by deadline.
2. Initialize current_time = 0.
3. For each process:
    a. If (current_time + burst_time <= deadline):
        i. Execute process.
        ii. Update current_time.
    b. Else:
        i. Print "Missed deadline."
```

References

Silberschatz, A., Galvin, P. B., & Gagne, G.
Operating System Concepts (10th Edition). Wiley.

- Covers fundamental and advanced CPU scheduling algorithms.

Linux Kernel Documentation – CPU Scheduling

- Available at: https://www.kernel.org/doc/html/latest/
- Covers process scheduling in Linux-based operating systems.

ACM Digital Library – Scheduling Algorithms

- Available at: https://dl.acm.org/
- Provides research on multi-level queue scheduling and reinforcement learning in scheduling.

Python Scikit-Learn – AI Model for Predictive Scheduling

- Available at: https://scikit-learn.org/stable/
- Provides machine learning tools for predictive scheduling.

OpenAI – ChatGPT Documentation

https://openai.com/chatgpt

Scikit-Learn – Machine Learning for Scheduling

- https://scikit-learn.org/stable/
- Provides machine learning tools for predictive scheduling models