ADDIS ABABA SCIENCE AND TECHNOLOGY UNIVERSITY

COLLEGE OF ENGINEERING

DEPARTMENT OF SOFTWARE ENGINEERING

Simulation and Modeling: Mini Project

Comparative Study of CPU Scheduling Algorithms Using Discrete-Event Simulation

Group

No.	Name	ID Number
1	Mikias Goitom	ETS1080/14
2	Natnael Fisseha	ETS1231/14
3	Natnael Necho	ETS1266/14
4	Tamirat Dejene	ETS1518/14
5	Temesgen Abebayehu	ETS1534/14

Inst.: Dr. Tesfay G.

Signature: _____

December 15, 2025

Contents

1. Introduction

CPU scheduling is a core component of operating systems (OS), responsible for allocating the central processing unit (CPU) to multiple processes vying for execution. Efficient scheduling aims to balance system throughput, resource utilization, and user responsiveness; however, poor scheduling decisions can lead to performance bottlenecks, process starvation, or under-utilization of system resources.

This project focuses on a comparative analysis of four classical CPU scheduling algorithms:

- First Come First Served (FCFS) — non-preemptive
- Shortest Job First (SJF-NP) — non-preemptive
- Priority Scheduling — non-preemptive
- Round Robin (RR) — preemptive

The importance of simulation in this context is significant. Analytical approaches such as queuing theory models (e.g., G/G/1) typically assume steady-state conditions and often fail to capture dynamic and stochastic behaviors, including random job arrivals and variable CPU burst times. Discrete-Event Simulation (DES) enables the modeling of such *"what-if"* scenarios over time, allowing artificial system histories to be generated and analyzed without disrupting real-world systems.

1.1 Project Objectives

The main objectives of this project are:

- To implement and compare the performance of FCFS, SJF-NP, Priority-NP, and Round Robin scheduling algorithms under stochastic workloads.
- To evaluate key performance metrics including Average Waiting Time (AWT), Average Turnaround Time (ATT), CPU Utilization, and Throughput.
- To analyze the sensitivity of scheduling performance to system parameters such as workload intensity (inter-arrival time) and scheduling overhead (context switch time).

1.2 Scope

This study models a single-core CPU system with stochastic inputs, where process inter-arrival times follow an Exponential distribution and CPU burst times follow a Normal distribution. The scope excludes multi-core processing, I/O interruptions, memory constraints, and secondary storage effects. To ensure statistical validity, the simulation is executed across multiple independent replications, aligning with standard software engineering practices for performance evaluation.

2. Problem Definition

The system under study is a single-server queuing model that abstracts the behavior of a Central Processing Unit (CPU) in a time-shared operating system. Formally, the system is represented as a G/G/1 queuing model, where:

- **Server:** The CPU, modeled as a single resource that can exist in either a *Busy* or *Idle* state.
- **Entities:** Processes (workloads) that arrive into the system and require a specified amount of CPU service time (burst time).
- **Queue:** The Ready Queue, which holds arriving processes waiting to be executed by the CPU. The queue capacity is assumed to be infinite.

The primary objective of the experiment is to model the dynamic state transitions of this system through discrete events governed by the selected scheduling algorithms. Particular attention is given to the time processes spend in the waiting state compared to the running state.

2.1 Real World Scenario

The simulation represents a high-contention, general-purpose computing environment similar to a busy server, virtual machine, or multi-tasking desktop operating system. In such an environment:

- **Stochastic Arrivals:** Jobs arrive randomly, reflecting user interactions or network-generated requests.
- **Variable Demands:** Each job requires a different amount of CPU processing time, ranging from short interactive commands to long-running computational tasks.
- **Contention:** The arrival rate is sufficiently high relative to the service rate, ensuring the ready queue is frequently populated and forcing the scheduler to make performance-critical decisions.

The real-world challenge addressed by this study is the selection of a scheduling strategy that maximizes efficient utilization of the CPU while minimizing excessive delays, particularly for short or interactive processes. This problem is exemplified by the *convoy effect* inherent in FCFS scheduling, where long processes delay all subsequent jobs.

2.2 Key Assumptions and Constraints

To maintain a focused and tractable simulation model, the following assumptions and constraints are applied:

- **Stochastic Input Fidelity:** Process inter-arrival times follow an Exponential distribution, and CPU burst times follow a Normal distribution to reflect realistic workload variability.
- **Context Switch Overhead:** A fixed context switch time is explicitly modeled and added to the simulation clock, enabling accurate evaluation of preemptive algorithms such as Round Robin.
- **No I/O Operations:** All processes are assumed to be CPU-bound; I/O blocking and waiting are excluded from the model.
- **Known Burst Time (SJF-NP):** The total CPU burst time of a process is assumed to be known at arrival for implementing the Shortest Job First algorithm.
- **Single-Core CPU:** The system is limited to a single CPU, eliminating parallel execution.
- **Initial State:** All simulations begin at time $t = 0$ with an empty ready queue.

3. Conceptual Model

In this section, we outline the conceptual model of the CPU scheduling simulation, including the system components, their attributes, and the interactions governing system behavior within the Discrete-Event Simulation (DES) framework.

3.1 Model Structure

3.1.1 The Queuing System

The system is architecturally based on a single-server queuing model. The overall state of the system is determined by the interaction between processes, the CPU resource, and the ready queue. The major components of the simulation model are summarized in Table 3.1.

Table 3.1: Core Components of the Simulation Model

Component	**Role in Simulation**
Entities	Processes (jobs) that require CPU service.
Resource	The CPU, modeled as a single server that executes one process at a time.
Source	Generates processes with stochastic arrival times and attributes.
Queues	The ready queue that holds processes waiting for CPU allocation.

3.1.2 High-Level Simulation Design

At a high level, the simulation consists of a process source feeding jobs into a ready queue, a scheduling mechanism selecting processes based on a chosen algorithm, and a single CPU resource executing the selected process. A conceptual overview of this interaction is illustrated in Figure 3.1.

3.1.3 Entities and Attributes

The fundamental entity in the simulation is the *Process Control Block (PCB)*, which encapsulates all information required for scheduling decisions and performance metric calculations. Each PCB contains the attributes listed in Table 3.2.

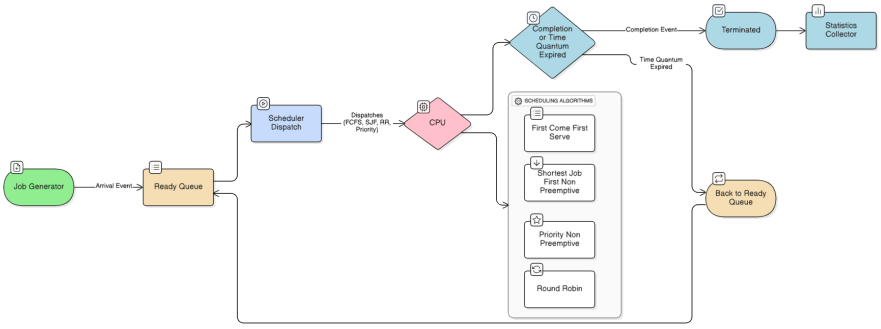

Figure 3.1: High-Level Simulation Design

Table 3.2: Process Control Block (PCB) Attributes

Attribute	**Description**	**Relevance to Scheduling**
PID	Unique process identifier.	Enables per-process tracking and metric computation.
Arrival Time	Simulation time at which the process enters the ready queue.	Determines scheduling eligibility.
Burst Time	Original total CPU time required by the process.	Used by SJF-NP and final metric calculations.
Remaining Burst Time	CPU time left until completion.	Required for preemptive scheduling in Round Robin.
Priority	Integer priority value (lower value indicates higher priority).	Used by the Priority-NP scheduling algorithm.

3.2 Simulation State Transition

The simulation progresses through a series of discrete state transitions driven by events such as process arrivals, context switches, and process completions. These transitions define how the system evolves over simulated time. A state transition diagram illustrating this behavior is shown in Figure 3.2.

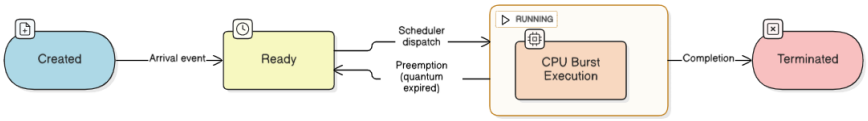

Figure 3.2: Simulation State Transition Diagram

3.3 The Discrete-Event Logic and Process Flow

The simulation operates using an event-driven loop that advances a global simulation clock and processes events in chronological order. While the scheduling decision logic differs between algorithms, the overall flow remains consistent across all experiments. The generalized

simulation process is described below:

- **Workload Setup:** A list of PCBs is generated, each with stochastic arrival time, CPU burst time, and priority values.
- **Initial Sorting:** The workload is sorted by arrival time to ensure controlled admission of processes into the simulation timeline.
- **Main Simulation Loop:** The simulation clock advances by processing the current event, incorporating the following decisions:
  - **Idle Time Handling:** If the arrival time of the next process is greater than the current clock value, the CPU remains idle and the clock is advanced directly to the next arrival time.
  - **Scheduling Decision:** The scheduler selects the most eligible process from the ready queue based on the rules of the active scheduling algorithm (e.g., shortest burst time for SJF-NP).
  - **Context Switch Penalty:** If the selected process differs from the previously executed process, a fixed context switch time is added to the simulation clock.
  - **Execution:** The clock is advanced by either the full burst time (for FCFS, SJF-NP, and Priority-NP) or by the time quantum (for Round Robin).
  - **New Arrivals:** All processes whose arrival times are less than or equal to the updated clock value are transferred into the ready queue.

This discrete-event approach ensures that all time-dependent costs, scheduling decisions, and system state changes are accurately modeled. As a result, performance metrics such as waiting time, turnaround time, CPU utilization, and throughput are computed correctly and consistently.

3.4 Simulation Flowchart

A flowchart summarizing the complete simulation logic and control flow is provided in Figure 3.3.

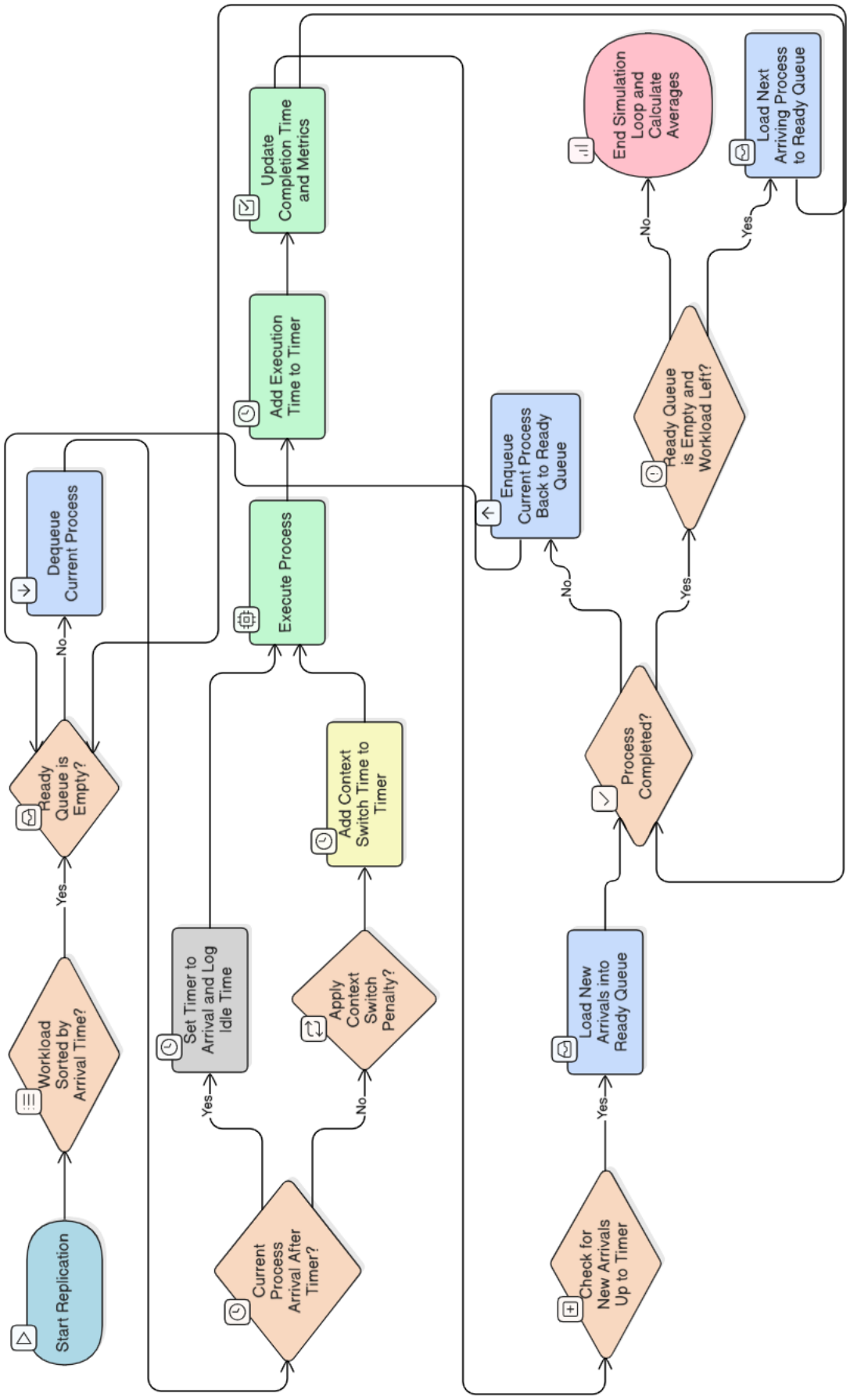

Figure 3.3: Simulation Flowchart

4. Data Collection and Input Analysis

In this section, we describe the process used to generate the synthetic workload that drives the simulation, justify the choice of statistical distributions, and explain the methods used to ensure that the generated input data reflects a realistic computing environment. Since real-world workload traces from production systems are often proprietary and difficult to obtain, we employ stochastic modeling to create a statistically representative workload.

4.1 Input Variables and Stochastic Modeling

Each simulated process is characterized by three primary stochastic input variables: inter-arrival time, CPU burst time, and priority. These variables are generated using well-established probability distributions commonly used in queuing theory and performance modeling.

4.1.1 Inter-Arrival Time (IAT)

The inter-arrival time represents the elapsed time between the arrivals of successive processes and is a primary determinant of system load and contention.

Chosen Distribution: Exponential Distribution (Poisson arrival process)

Rationale: The Exponential distribution is the standard model for inter-arrival times in classical queuing systems such as the M/G/1 model. It assumes arrivals occur independently and continuously at a constant average rate, which closely matches the random nature of job submissions in time-shared operating systems.

Implementation: Inter-arrival times are generated using the inverse transform method:

$$t = -\lambda \ln(1 - U)$$

where λ is the mean inter-arrival time and U is a uniformly distributed random variable in the range $[0, 1)$.

4.1.2 CPU Burst Time (Service Demand)

CPU burst time represents the total processing time required by a process and varies significantly between different workloads, ranging from short interactive tasks to long-running computational jobs.

Chosen Distribution: Normal (Gaussian) Distribution

Rationale: Although real-world CPU burst times often exhibit heavy-tailed behavior, the Normal distribution provides a statistically controlled means of modeling a majority of jobs clustered around a mean value (μ) while allowing variability through a standard deviation (σ). This approach facilitates clearer comparative analysis between scheduling algorithms.

Implementation: Burst times are generated using standard library functions that transform uniform random variates into a Normal distribution with specified mean μ and standard deviation σ.

4.1.3 Process Priority

Process priority is required for implementing the Priority (non-preemptive) scheduling algorithm.

Chosen Distribution: Uniform Distribution (Discrete Integer)

Rationale: A uniform distribution over a fixed priority range (e.g., 1 to 5) ensures that all priority levels are equally likely. This prevents systematic bias toward any specific priority class and enables fair performance comparison.

4.2 Data Generation and Preprocessing Steps

The workload generation procedure is encapsulated within the `JobGenerator` class and is executed prior to each simulation replication.

- **Workload Size:** The total number of jobs (N) per replication is fixed using predefined hyperparameters (e.g., $N = 100$ jobs).
- **Generation:** For each of the N processes, inter-arrival time, CPU burst time, and priority are generated using the distributions described above.
- **Data Structure:** The generated attributes are encapsulated into a list of Process Control Block (PCB) objects.
- **Preprocessing (Deep Copying):** Prior to executing each scheduling algorithm (FCFS, SJF-NP, Priority-NP, and RR), the `ExperimentManager` creates a deep copy of the original PCB list. This ensures that all algorithms operate on an identical workload within each replication, isolating observed performance differences strictly to scheduling logic.

4.3 Analysis of Input Data for Experimentation

Input parameters are systematically controlled across multiple simulation scenarios to evaluate algorithm robustness under varying operating conditions. Key parameters manipulated during experimentation are summarized in Table 4.1.

By varying the mean inter-arrival time λ while keeping the CPU burst time mean μ constant, controlled experimental environments are created. This approach enables systematic stress

Table 4.1: Controlled Input Parameters for Experimentation

Scenario Parameter	**Purpose in Experimentation**
Mean Inter-Arrival Time (λ)	Controls system load. Smaller values (e.g., 2.0 ms) produce high contention, while larger values (e.g., 10.0 ms) produce low contention.
Context Switch Time	Varied from 0 ms to 5 ms to quantify the overhead penalty introduced by context switching, particularly affecting preemptive algorithms such as Round Robin.
Time Quantum (RR)	Fixed to evaluate the impact of preemption frequency on waiting time, turnaround time, and CPU utilization.

testing of scheduling algorithms and supports valid conclusions regarding their performance and real-world applicability.

5. Simulation Design

This section outlines the simulation technique employed, the software tools and programming environment, and the design principles used to implement the comparative CPU scheduling simulation model.

5.1 Simulation Technique

The simulation model is implemented using the *Discrete-Event Simulation (DES)* technique.

Rationale: DES is the most appropriate approach for modeling queuing-based systems such as CPU scheduling. In this system, the state (e.g., CPU utilization, contents of the ready queue) changes only at discrete, well-defined points in time corresponding to events such as process arrival, process completion, or time quantum expiration. Continuous simulation is unnecessary for such systems, while Monte Carlo simulation is more suitable for static probabilistic analysis rather than dynamic processes evolving over time.

Mechanism: The DES engine operates using a global simulation clock that advances non-contiguously from one event time to the next. Idle periods are skipped entirely, making the simulation computationally efficient while accurately modeling the instantaneous nature of scheduling decisions.

5.2 Software Tools and Programming Environment

A two-tier toolchain was adopted, separating simulation modeling from statistical analysis and reporting.

5.2.1 Core Simulation Engine

Tool: Java (JDK 17+)

Rationale: Java was chosen for its strong object-oriented programming (OOP) support, enabling a clean, modular, and extensible design, as well as the team's familiarity with the language.

- **Inheritance:** A base `Scheduler` class encapsulates shared logic and metric collection, while specific scheduling algorithms (FCFS, Round Robin, Priority Scheduling, and Shortest Job First) are implemented as subclasses. This structure promotes code reuse, clarity, and easier verification.

- **Data Structures:** Java's standard collections framework (`LinkedList`, `PriorityQueue`, `HashMap`) is used to efficiently implement ready queues, event handling, and metric storage.

5.2.2 Statistical Analysis and Reporting

Tool: Python (Pandas, Matplotlib, Seaborn)

Rationale: The Java simulation exports raw experimental results—including waiting time, turnaround time, CPU utilization, and throughput for each replication—into CSV files. Python is used as a post-processing and visualization platform to:

- Aggregate data across multiple replications.
- Compute descriptive statistics such as mean and standard deviation.
- Generate 95% confidence intervals using the Student's t-distribution (as coordinated by the `ExperimentManager`).
- Produce high-quality comparative plots for result interpretation and reporting.

5.3 Design and Implementation of the Simulation Model

The simulation model emphasizes modularity, statistical control, and accurate performance metric collection.

5.3.1 Modular Design

The implementation is organized into the following core modules:

- **PCB:** Represents the Process Control Block, serving as the data container for process attributes.
- **JobGenerator:** Responsible for stochastic workload generation using Exponential and Normal distributions.
- **Scheduler Hierarchy:** Contains the scheduling logic and metric calculations for each algorithm.
- **ExperimentManager:** Acts as the control module that orchestrates simulation execution and experimentation.

5.3.2 The ExperimentManager: Statistical Control

The `ExperimentManager` enforces statistical rigor throughout the experimentation process:

- Defines uniform hyperparameters (like the number of jobs, mean inter-arrival time, context switch time).

- Ensures that 100 (N) independent replications are executed for each scheduling algorithm and scenario.
- Guarantees that, for each replication, all scheduling algorithms receive an identical initial workload by creating deep copies of the original PCB list. This isolates observed performance differences strictly to scheduling policy behavior.

5.3.3 Core Performance Metrics (Output Variables)

The simulation computes the following performance metrics for every completed process. These metrics are averaged per replication and later aggregated across replications.

Table 5.1: Core Performance Metrics

Metric	**Definition**	**Purpose**
Average Waiting Time ($\bar{WT}$)	Time a process spends waiting in the ready queue.	Measures system responsiveness and fairness.
Average Turnaround Time ($\bar{TAT}$)	Time from process arrival to completion.	Measures overall job execution efficiency.
CPU Utilization (%)	Ratio of CPU busy time to total simulation time.	Measures resource efficiency.
Throughput (jobs/ms)	Number of completed processes per unit time.	Measures system capacity and productivity.

The metrics are computed using formulas implemented in the `Scheduler.calculateExtendedMetrics()` method:

$$\text{CPU Utilization} = \frac{\sum \text{Original Burst Time}}{\text{Total Simulation Time}} \times 100$$

$$\text{Throughput} = \frac{\text{Total Processes Completed}}{\text{Total Simulation Time}}$$

6. Model Verification and Validation

This section establishes the credibility of the simulation model by demonstrating that the implementation correctly reflects the conceptual design (*verification*) and that the observed output behavior reasonably represents the expected performance of real-world CPU scheduling systems (*validation*).

6.1 Verification: Correct Implementation of the Model

Verification focuses on confirming that the programmed logic is free from implementation errors and accurately reflects the intended scheduling algorithms and discrete-event mechanisms.

6.1.1 Deterministic Tracing and Manual Checks

The primary verification technique employed was deterministic tracing, which involved the following steps:

- **Small, Fixed Workload:** The simulation was executed using a small, manually constructed workload consisting of 5–10 processes with predefined arrival times, CPU burst times, and priority values.
- **Gantt Chart Output:** The detailed execution schedule (schedule table or *Gantt chart*) produced by the `Scheduler` class was examined.
- **Manual Calculation:** Expected completion time, turnaround time, and waiting time values were calculated manually according to the rules of FCFS, SJF-NP, and Round Robin scheduling.
- **Comparison:** The manually derived values were compared directly with the metrics produced by the simulation.

This verification process confirmed the correctness of the following key logic components:

- **Clock Advancement:** The global simulation clock increases correctly based on CPU burst execution, idle time, and context switch overhead.
- **Queue Prioritization:** Priority queue implementations correctly select the shortest remaining job for SJF-NP and the highest-priority job for Priority-NP.
- **Context Switch Logic:** The context switch time is applied exactly once per process switch in the Round Robin implementation.

6.1.2 Code Review and Modular Testing

The modular design of the implementation, with separate classes for each scheduling algorithm, enabled targeted code review and testing. Particular attention was given to the deep-copy mechanism implemented in the `ExperimentManager`, ensuring that each scheduling algorithm begins execution with an identical and unmodified workload. This guarantees that observed performance differences arise solely from scheduling logic rather than input variation.

6.2 Validation: Representing Real-World Behavior

Validation ensures that the simulation output exhibits behavior consistent with established operating system theory and queuing models.

6.2.1 Comparison to Theoretical Expectations

The simulation results were validated by confirming alignment with well-known theoretical properties of CPU scheduling algorithms, particularly under high-contention conditions:

- **SJF-NP Optimality:** The Shortest Job First (non-preemptive) algorithm consistently produced the lowest Average Waiting Time among all non-preemptive schedulers, confirming its theoretical optimality.
- **FCFS Boundary Behavior:** First Come First Served scheduling consistently yielded the highest Average Waiting Time, reflecting its susceptibility to the *convoy effect*.
- **Preemption Overhead:** Introducing a non-zero context switch time resulted in a measurable reduction in CPU utilization for the Round Robin algorithm, as theoretically expected for preemptive systems.

6.2.2 Input Distribution Check

Although the workload data is synthetically generated, its adherence to the selected statistical distributions is essential for external validity. Histograms of the generated inter-arrival times and CPU burst times were visually inspected and confirmed to match the characteristic shapes of the Exponential and Normal distributions, respectively.

6.2.3 Statistical Stability

Statistical stability was verified by evaluating the variance of the simulation outputs. The use of 100 independent replications, combined with the resulting narrow 95% confidence intervals, demonstrates that the simulation reached steady-state behavior and that the estimated mean performance metrics are reliable.

7. Experimentation

This section describes the experimental design and execution plan for the comparative study. Using the validated Discrete-Event Simulation (DES) model, the four CPU scheduling algorithms are rigorously evaluated under distinct and representative operating conditions. The experimental design focuses on isolating the effects of system load and preemption overhead on key performance metrics.

7.1 Experimental Design and Methodology

To ensure statistical validity and a fair comparison across scheduling algorithms, the following methodology is consistently applied across all experimental scenarios:

- **Replication Strategy:** The simulation is executed for 1000 independent replications for each scheduling algorithm in every scenario. This large number of replications ensures reliable estimation of mean performance values and the construction of tight 95% confidence intervals (CI), minimizing stochastic noise.
- **Controlled Workload:** For a given replication, all four algorithms (FCFS, SJF-NP, Priority-NP, and Round Robin) are executed using an identical stochastically generated workload (deep-copied PCB list). This isolates performance differences strictly to scheduling logic.
- **Performance Metrics:** For each replication, the recorded outputs are the average Waiting Time (WT), Turnaround Time (TAT), CPU Utilization, and Throughput.

7.2 Experimental Scenarios (Treatments)

The experimentation is divided into three distinct scenarios designed to evaluate algorithm behavior under varying system load and operational realism. The mean CPU burst time is held constant across all scenarios to isolate the impact of the arrival rate (λ).

7.2.1 Core Workload Parameters (Constant)

The workload parameters listed in Table 7.2 are held constant across all scenarios to ensure experimental control.

Table 7.1: Experimental Scenarios

Scenario	**Objective**	**Mean IAT (λ)**	**Load Type**	**Context Switch Time**
Scenario 1: Low Contention	Baseline performance with minimal queue buildup	10.0 ms	Underutilized	0 ms
Scenario 2: High Contention	Stress-test algorithms to maximize performance differences	2.0 ms	Saturated	0 ms
Scenario 3: Realistic Load	Quantify preemption overhead in practical environments	5.0 ms	Moderate / High	1.0 ms

Table 7.2: Constant Workload Parameters

Parameter	**Value**	**Distribution**
Number of Jobs per Replication	100	N/A
Mean CPU Burst Time (μ)	8.0 ms	Normal
Standard Deviation (σ)	2.0 ms	Normal
Maximum Priority Level	10 (Lower = Higher Priority)	Uniform
Time Quantum (RR)	5.0 ms	N/A

7.2.2 Analysis of Scenario 2 (High Contention)

Scenario 2 represents the primary comparative evaluation phase. By setting the mean inter-arrival time to $\lambda = 2.0$ ms—significantly lower than the mean CPU burst time of 8.0 ms—the system rapidly becomes saturated. As a result, the ready queue grows continuously, forcing the scheduler to repeatedly select processes from a large pool of waiting jobs.

Expected Outcome: Under these conditions, performance metrics such as Average Waiting Time ($\bar{WT}$) and Average Turnaround Time ($\bar{TAT}$) are expected to diverge maximally. The SJF-NP algorithm is expected to demonstrate superior performance by minimizing $\bar{WT}$, while FCFS is expected to perform worst due to the convoy effect.

7.2.3 Analysis of Scenario 3 (Realistic Load)

Scenario 3 introduces two key real-world considerations: a moderate system load ($\lambda = 5.0$ ms) and a non-zero context switch time of 1.0 ms.

Objective: To quantify the inherent trade-off of preemptive scheduling. In Round Robin scheduling, frequent preemption leads to repeated context switches, consuming CPU time that could otherwise be devoted to useful computation.

Expected Outcome: The 1.0 ms context switch overhead is expected to produce a statistically significant reduction in CPU Utilization and Throughput for the Round Robin algorithm when compared to non-preemptive algorithms (FCFS, SJF-NP, Priority-NP) under the same workload conditions.

7.3 Data Recording and Analysis

The `ExperimentManager` module coordinates execution of all experimental runs and manages result aggregation for analysis.

7.3.1 Raw Data Export

Results from all replications across the 3×4 experiment combinations are consolidated into a single CSV file. Each record includes the replication index, scheduling algorithm, performance metric type, and measured value, enabling efficient post-processing using Python and Pandas.

7.3.2 Statistical Processing

The final reported results are computed using the following statistical procedures:

- **Mean Calculation:** The sample mean ($\bar{X}$) is calculated for each metric across all replications.
- **Standard Deviation:** The sample standard deviation (σ) is computed to measure dispersion.
- **Confidence Interval (CI):** A 95% confidence interval is calculated using:

$$\text{CI} = \bar{X} \pm \left(t_{\alpha/2,\, n-1} \times \frac{\sigma}{\sqrt{n}} \right)$$

For $n = 100$ replications, the critical value $t_{\alpha/2,n-1}$ is approximately 1.984, which closely approximates the standard normal value of 1.96 for a 95% confidence level. This margin of error quantifies the statistical reliability of the reported results.

8. Results and Analysis of Realistic Load with Overhead

This section presents and interprets the results obtained from **Scenario 3: Realistic Load with Overhead** ($\lambda = 5.0$ ms, Context Switch Time = 1.0 ms). The objective of this scenario is to analyze the performance trade-offs between preemptive and non-preemptive scheduling algorithms under moderate-to-high system load while incorporating realistic context switch costs. The analysis is based on the generated graphs and aggregated statistical results.

8.1 Comparative Performance Overview

Figures 8.1, 8.2, 8.3, and 8.4 summarize the comparative behavior of the four scheduling algorithms in terms of responsiveness and efficiency metrics.

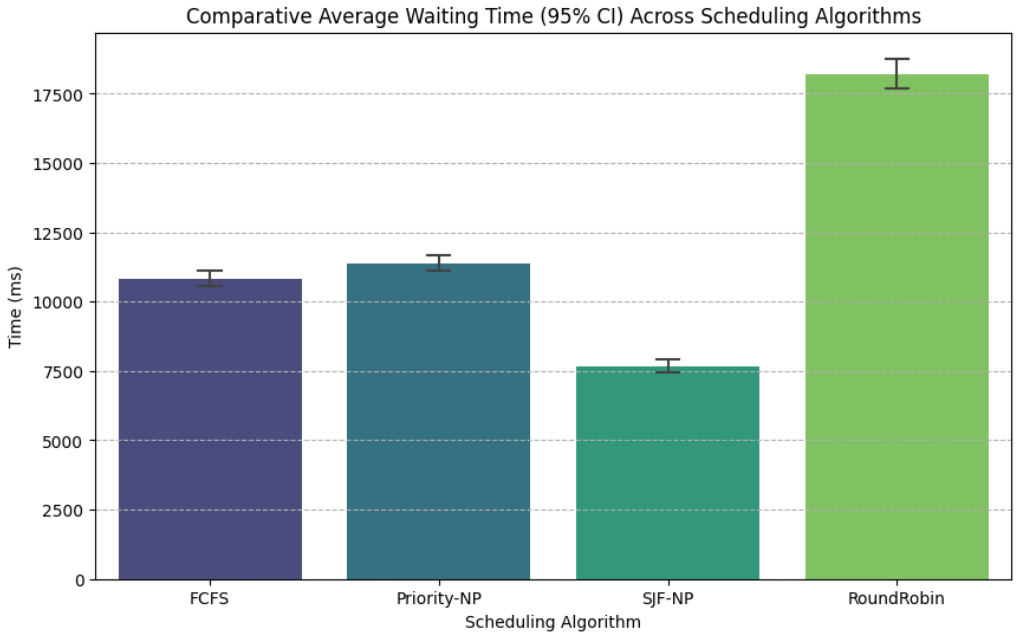

Figure 8.1: Comparative Average Waiting Time

8.2 Responsiveness and Latency Analysis ($\bar{WT}$ and $\bar{TAT}$)

Responsiveness metrics—Average Waiting Time ($\bar{WT}$) and Average Turnaround Time ($\bar{TAT}$) —measure the latency experienced by processes before and during execution. Lower values indicate superior scheduling performance.

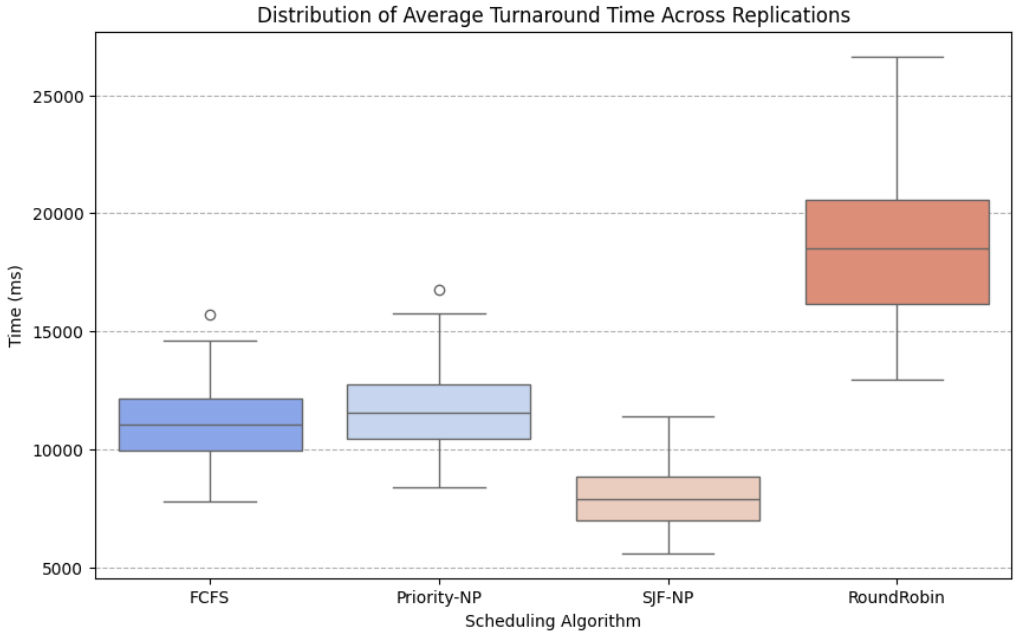

Figure 8.2: Comparative Average Turnaround Time

8.2.1 SJF-NP Optimality

As shown in Figure 8.1, the Shortest Job First Non-Preemptive (SJF-NP) algorithm achieved the lowest Average Waiting Time among all algorithms. The mean $\bar{WT}$ for SJF-NP was approximately 7,678 ms, confirming its theoretical optimality in minimizing waiting time under known burst durations. This behavior is consistent with scheduling theory, which states that SJF minimizes average waiting time in non-preemptive systems.

8.2.2 Round Robin Latency Penalty

In contrast, the Round Robin (RR) algorithm exhibited the highest latency. The mean $\bar{WT}$ for RR was approximately 18,220 ms, as illustrated in Figure 8.1. This observation is further reinforced by the Average Turnaround Time distribution shown in Figure 8.2, where the RR box plot is positioned highest on the time axis, with a mean $\bar{TAT}$ of approximately 18,449 ms.

This elevated latency reflects the combined impact of frequent preemptions, repeated context switching, and prolonged queue residence times under moderate-to-high load conditions.

8.3 Efficiency and Overhead Analysis (CPU Utilization and Throughput)

Efficiency metrics evaluate how effectively the CPU is used for productive work. In this scenario, efficiency is heavily influenced by the introduction of a 1.0 ms context switch overhead.

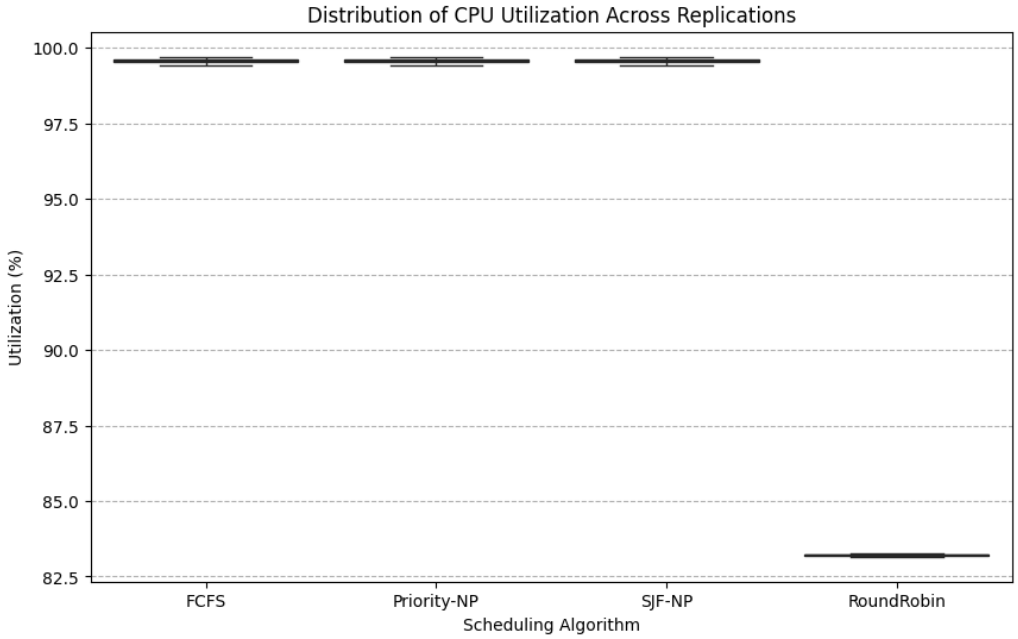

Figure 8.3: CPU Utilization Across Replications

8.3.1 Quantifying the Cost of Preemption

The CPU Utilization results shown in Figure 8.3 provide the clearest evidence of preemption overhead:

- The non-preemptive algorithms (FCFS, Priority-NP, and SJF-NP) maintained near-perfect utilization, each averaging approximately 99.56%.
- The Round Robin algorithm experienced a substantial reduction in utilization, with a mean value of approximately 83.21%.

This represents a **16.35% absolute loss in CPU utilization**, which directly corresponds to the repeated application of the 1.0 ms context switch penalty at every 5.0 ms time quantum. The result quantitatively validates the theoretical expectation that preemption introduces non-trivial overhead in practical systems.

8.3.2 Impact on System Capacity (Throughput)

The Throughput results further confirm the efficiency degradation observed in Round Robin scheduling. As shown in Figure 8.4:

- The non-preemptive algorithms achieved comparable throughput values of approximately 0.0018 jobs/ms.
- Round Robin's throughput was significantly lower, at approximately 0.0001 jobs/ms.

The reduction in throughput reflects the diminished effective CPU time available for executing jobs due to frequent context switches, thereby lowering the system's overall processing capacity.

	Algorithm	Metric	mean	median	std	\
0	FCFS	AvgTurnaroundTime	11067.2966	11062.065	1489.336702	
1	FCFS	AvgWaitingTime	10837.9315	10833.575	1460.043850	
2	FCFS	CPUUtilization	99.5628	99.570	0.058207	
3	FCFS	Throughput	0.0018	0.000	0.003861	
4	Priority-NP	AvgTurnaroundTime	11611.0520	11559.085	1572.255414	
5	Priority-NP	AvgWaitingTime	11381.6869	11326.255	1542.943881	
6	Priority-NP	CPUUtilization	99.5628	99.570	0.058207	
7	Priority-NP	Throughput	0.0018	0.000	0.003861	
8	RoundRobin	AvgTurnaroundTime	18449.3785	18497.545	2772.590952	
9	RoundRobin	AvgWaitingTime	18220.0134	18263.970	2743.397709	
10	RoundRobin	CPUUtilization	83.2112	83.210	0.020463	
11	RoundRobin	Throughput	0.0001	0.000	0.001000	
12	SJF-NP	AvgTurnaroundTime	7907.2694	7898.695	1170.859401	
13	SJF-NP	AvgWaitingTime	7677.9043	7665.185	1141.694332	
14	SJF-NP	CPUUtilization	99.5628	99.570	0.058207	
15	SJF-NP	Throughput	0.0018	0.000	0.003861	

Figure 8.4: Grouped Descriptive Statistics by Algorithm and Metric

8.4 Summary of Findings

Under realistic operating conditions with moderate load and non-zero context switch overhead, non-preemptive algorithms demonstrated superior efficiency and responsiveness. While Round Robin offers fairness through time slicing, this scenario clearly illustrates that such fairness comes at a measurable cost in latency, CPU utilization, and throughput. These findings reinforce the importance of selecting scheduling algorithms based on workload characteristics and operational constraints rather than theoretical fairness alone.

9. Conclusion

9.1 Summary of Findings and Significance

This study employed a Discrete-Event Simulation (DES) framework to rigorously evaluate and compare the performance of four CPU scheduling algorithms—FCFS, SJF-NP, Priority-NP, and Round Robin—under a realistic operating environment characterized by moderate-to-high contention ($\lambda = 5.0$ ms) and a non-zero context switch overhead of 1.0 ms.

The results clearly demonstrate that **Shortest Job First (Non-Preemptive)** is the most effective algorithm for minimizing process delay. Across all replications, SJF-NP consistently achieved the lowest Average Waiting Time ($\bar{WT}$) and Average Turnaround Time ($\bar{TAT}$), confirming its well-established theoretical optimality for latency minimization in single-server systems.

The most significant insight from this study is the **quantification of preemption overhead**. The introduction of a 1.0 ms context switch cost caused the Round Robin (RR) scheduler's CPU Utilization to drop sharply from the non-preemptive baseline of 99.56% to only **83.21**%. This represents a substantial **16.35**% loss in effective CPU time, directly attributable to frequent preemption and context switching. The reduction in utilization was further reflected in a marked decrease in system throughput, highlighting the inefficiency of RR in throughput-sensitive environments.

Overall, the findings strongly support the use of non-preemptive scheduling strategies—particularly SJF-NP—in server or batch-processing systems where minimizing latency and maximizing resource efficiency are primary objectives, and where reasonable estimates of process execution time are available.

9.2 Limitations of the Study

While the simulation model provides valuable insights, several limitations constrain the generalizability of the results:

- **Known Burst Time Assumption:** The implementation of SJF-NP assumes exact knowledge of CPU burst times at process arrival. In real operating systems, burst times must be estimated, which may reduce the observed performance advantage of SJF-based algorithms.
- **Absence of I/O Operations:** The model exclusively considers CPU-bound processes

and does not account for I/O blocking behavior. In practical systems, I/O-bound processes frequently relinquish the CPU, altering ready queue dynamics and scheduling outcomes.

- **Single-Core Abstraction:** The simulation is limited to a single-core CPU model. Modern multi-core (SMP) systems introduce additional complexities such as load balancing, cache affinity, and concurrent queue access, which are not captured in this study.

9.3 Recommendations for Future Work

To enhance realism and extend the applicability of the model, future work should consider the following directions:

- **Burst Time Estimation:** Implement the Shortest Remaining Time First (SRTF) algorithm using exponential averaging to estimate future CPU bursts, thereby approximating real-world OS scheduling behavior.
- **I/O Behavior Modeling:** Introduce probabilistic I/O blocking to allow processes to transition between Ready and I/O Waiting states, enabling the evaluation of mixed CPU- and I/O-bound workloads.
- **Advanced Scheduling Policies:** Extend the comparative study to include production-level schedulers such as the Multi-Level Feedback Queue (MLFQ), which dynamically adapts priorities to balance fairness, responsiveness, and efficiency without requiring prior knowledge of burst times.

10. Documentation and Appendices

This section provides comprehensive documentation required to understand, replicate, and extend the CPU Scheduling Discrete-Event Simulation (DES) model. It includes the source code organization, raw data management strategy, and a concise user manual for executing the simulation and reproducing the experimental results.

10.1 Source Code and Project Structure

The simulation model was implemented in Java using a modular, object-oriented design to ensure extensibility, maintainability, and verification transparency.

10.1.1 Project Directory Structure

```
+-- src.process.scheduler/
|   +-- DiscreteEventSimulator.java   // Main entry point and scenario init
|   +-- ExperimentManager.java        // Controls replications, params
|   +-- PCB.java                      // Process Control Block (entity defn)
|   +-- JobGenerator.java             // Stochastic workload generation
|   +-- Scheduler.java                // Abstract base scheduler class
|   +-- FCFS.java                     // First-Come, First-Served impl.
|   +-- PriorityScheduling.java       // Non-preemptive Priority scheduling
|   +-- ShortestJobFirst.java         // Non-preemptive SJF scheduling
|   \-- RoundRobin.java               // Preemptive Round Robin scheduling
|
+-- simulation_results.csv        // Raw simulation output (all replications)
+-- analysis.py                   // Python analysis and visualization script
+-- assets/                      // Generated charts and plots
+-- docs/                         // Project report and documentation
\-- README.md                         // Project overview and execution inst.
```

10.1.2 Key Code Component: Experiment Manager Hyperparameters

The `ExperimentManager` defines and enforces the hyperparameters controlling each experimental scenario. These parameters ensure consistent workload generation and statistical rigor across replications.

```
public class ExperimentManager {

    public static class Parameters {
        public final int numJobs;
        public final double meanInterArrival;
        public final double meanBurst;
        public final double stdDevBurst;
        public final int maxPriority;
        public final int timeQuantum;
        public final double contextSwitchTime;
        public final int replications;

        public Parameters(int numJobs, double meanInterArrival,
                          double meanBurst, double stdDevBurst,
                          int maxPriority, int timeQuantum,
                          double contextSwitchTime, int replications) {
            // Initializes all hyperparameters for a scenario
        }
    }
    // Logic for deep copying workloads, running replications,
    // and collecting performance metrics
}
```

10.2 Raw Data Management

To ensure statistical reliability and transparency, the simulation records all output metrics in a structured raw data format.

10.2.1 Data Format and Storage

- **Format:** Comma-Separated Values (CSV)
- **File Location:** `./simulation_results.csv`
- **Granularity:** Each row corresponds to one replication of one algorithm

10.2.2 Recorded Data Fields

Field Name	Description
Replication	Replication index (1 to N)
Algorithm	Scheduling algorithm used
AvgWaitingTime	Mean waiting time (ms)
AvgTurnaroundTime	Mean turnaround time (ms)
CPUUtilization	CPU utilization percentage
Throughput	Jobs completed per millisecond

Table 10.1: Raw data fields recorded per simulation replication

10.3 User Manual for Simulation Operation

This subsection describes how to execute the simulation and generate the experimental results.

10.3.1 Prerequisites

- Java Development Kit (JDK) version 17 or higher
- Java-compatible IDE/Editor (VS Code, IntelliJ, Eclipse, CMD tools, etc.)
- Python 3.x installed
- Jupyter Notebook or JupyterLab environment
- Required Python libraries: `pandas`, `matplotlib`, `seaborn`

10.3.2 Running the Simulation

1. **Locate the Main Class:**
   Navigate to `process.scheduler.DiscreteEventSimulator.java`.

2. **Select Experimental Scenario:**
   In the `main()` method, comment or uncomment the desired scenario configuration (`params1`, `params2`, or `params3`).

   ```
   // Example: Run only Scenario 3
   System.out.println("Starting SCENARIO 3: Realistic Load with Overhead");
   ExperimentManager manager3 = new ExperimentManager(params3);
   manager3.runComparativeStudy();
   ```

3. **Execute the Program:**
   Run the `DiscreteEventSimulator` class. The console will display statistical summaries (mean ± 95% CI) for all scheduling algorithms.

4. **Data Output:**
   Raw simulation results are automatically saved to `./simulation_results.csv`.

10.3.3 Post-Processing and Graph Generation

Post-processing and visualization of the simulation results were performed using a Jupyter Notebook environment.

1. Navigate to the `./` directory.
2. Launch Jupyter Notebook:

```
jupyter notebook
```

3. Open the notebook file (`analysis.ipynb`).
4. Execute the notebook cells sequentially to:
   - Load the `simulation_results.csv` file
   - Aggregate metrics across replications
   - Compute descriptive statistics and 95% confidence intervals
   - Generate comparative plots (AWT, TAT, CPU Utilization, Throughput)

The generated figures are displayed within the Jupyter Notebook environment and can be manually saved or exported by the user for inclusion in the final report.

Bibliography

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, Wiley, Latest Edition.

[2] W. Stallings, *Operating Systems: Internals and Design Principles*, Pearson Education, Latest Edition.

[3] A. M. Law, *Simulation Modeling and Analysis*, McGraw-Hill Education, Latest Edition.

[4] D. G. Kendall, "Stochastic Processes Occurring in the Growth of Bacterium and Their Associated Markov Chains," *The Annals of Mathematical Statistics*, vol. 24, no. 3, pp. 338–354, 1953.

[5] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris, *Fundamentals of Queueing Theory*, Wiley, Latest Edition.

[6] Oracle Corporation, *The Java Tutorials*, Latest Version.

[7] W. McKinney, *Python for Data Analysis*, O'Reilly Media, Latest Edition.

[8] M. Waskom, *Seaborn: Statistical Data Visualization*, Latest Version.