A

Project Report

On

PROCESS SCHEDULER

Submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

In

Computer Science and Engineering

By

Priya Koranga (2261433)

Chetna Karayat (2261157)

Kareena Aithani (2261313)

Saniya Chaniyal (2261511)

Under the Guidance of

Mr.Prince Kumar

ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS

SATTAL ROAD, P.O. BHOWALI,

DISTRICT- NAINITAL-263132

2024-2025

STUDENT’S DECLARATION

We, **Priya Koranga, Chetna Karayat, Kareena Aithani, Saniya Chaniyal** hereby declare the work, which is being presented in the project, entitled ‘**Process Scheduler**’ in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (B.Tech.)** in the session **2024-2025**, is an authentic record of my work carried out under the supervision of Mr. Prince Kumar..

The matter embodied in this project has not been submitted by me for the award of any other degree.

Date:

Priya Koranga

Chetna Karayat

Kareena Aithani

Saniya Chaniyal

CERTIFICATE

The project report entitled "Process Scheduler" being submitted by Priya Koranga, Chetna Karayat, Kareena Aithani, Saniya Chaniyal D/o Mr.Jagdish Singh koranga, Mr.Mahendra Singh Karayat, Himmat Singh Aithani, Mr. Bhupal Chaniyal, 2261433, 2261157, 2261313, 2261511 of B.Tech.(CSE) to Graphic Era Hill University Bhimtal Campus for the award of bonafide work carried out by them.They have worked under my guidance and supervision and fulfilled the requirement for the submission of a report.

Mr.Prince Kumar	**Dr. Ankur Singh Bisht**
(Project Guide)	**(Head, CSE)**

ACKNOWLEDGEMENT

We take immense pleasure in thanking the Honorable Director**‘Prof. (Col.)Anil Nair (Retd.)’,** GEHU Bhimtal Campusto permit me and carry out this project work with his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance, and useful suggestions that helped me to develop as a creative researcher and complete the research work, in time.

Words are inadequate in offering my thanks to GOD for providing me with everything that we need. We again want to extend thanks to our president**‘Prof. (Dr.) Kamal Ghanshala’** for providing us with all infrastructure and facilities to work in need without which this work could not be possible.

Many thanks to **‘Dr. Ankur Singh Bisht’** (Head, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project guide**‘Mr. Prince Kumar’**(Assistant Professor, Department of Computer Science and Engineering, GEHU Bhimtal Campus)and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this report.

Finally, yet importantly, We would like to express my heartiest thanks to our beloved parents, for their moral support, affection, and blessings. We would also like to pay our sincere thanks to all my friends and well-wishers for their help and wishes for the successful completion of this project.

Priya Koranga,2261433

Chetna Karayat,2261157

Kareena Aithani,2261313

Saniya Chaniyal,2261511

Abstract

This project presents the design and implementation of a **Process Scheduler**, a simulation tool developed to illustrate and compare the functionality of various CPU scheduling algorithms used in operating systems. The scheduler supports multiple scheduling strategies, including **First-Come, First-Served (FCFS)**, **Shortest Job First (SJF)**, **Priority Scheduling**, and **Round Robin (RR)**. These algorithms are fundamental to process management, determining the order in which processes are executed by the CPU to optimize performance.

The objective of this project is to provide a clear understanding of how different scheduling algorithms affect key performance metrics such as **waiting time**, **turnaround time**, **response time**, and **CPU utilization**. Users can input process details such as arrival time, burst time, and priority, and the system calculates and visually displays the scheduling order, Gantt charts, and relevant statistics for each algorithm.

This simulation aids in visualizing the internal working of a CPU scheduler, making it easier to grasp how theoretical scheduling principles are applied in practical scenarios. The system is capable of handling multiple processes and can simulate both preemptive and non-preemptive scheduling based on the selected algorithm.

The project is implemented using [Insert language/platform here], with a focus on user-friendliness, modular code structure, and performance efficiency. It also provides a comparative analysis of each algorithm’s output, allowing users to evaluate their effectiveness under different conditions.

Overall, this project serves as a valuable educational tool for students, educators, and system designers by providing a hands-on experience of CPU process scheduling, enhancing their understanding of process management in real-time operating systems.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

Abbreviation :

CU - Central Processing Unit**, GUI** - Graphical User Interface

RR – Round Robin, **FCFS** – First Come First Serve

SJF – Shortest Job First, **PID** – Process Identifier

I/O – Input/Output, **JVM** – Java Virtual Machine

API – Application Programming Interface, **ms** – Milliseconds

ns – Nanoseconds **FIFO** – First-In First-Out, **QoS** – Quality of Service

OOP – Object-Oriented Programming, **IDE** – Integrated Development Environment

CHAPTER 1 INTRODUCTION

1.1 Prologue

In the ever-evolving field of computer science, the operating system (OS) serves as the core software that manages hardware resources and provides essential services for the execution of application programs. Among its many critical tasks, **process scheduling** is one of the most fundamental. It determines the order in which processes access the CPU, especially in environments where multiple processes compete for limited computing resources.

Scheduling directly influences key aspects of system performance such as **CPU utilization, throughput, turnaround time, waiting time, and response time**. The objective of a good scheduling algorithm is to maximize resource usage while minimizing delays and ensuring fairness among processes. This becomes particularly important in multi-user, multitasking, or real-time operating systems where responsiveness and efficiency are crucial.

The concept of process scheduling is well-covered in academic theory, but learners often find it challenging to visualize how these algorithms work in practice. Most textbooks provide algorithmic descriptions and numerical examples, but lack interactive tools or simulations that can help users truly understand how a scheduler behaves in real time. As a result, there is a gap between theoretical understanding and practical insight.

This project, titled **"Process Scheduler"**, is developed to address this educational gap. It is a simulation-based system that demonstrates the functionality of several widely-used CPU scheduling algorithms, including **First-Come, First-Served (FCFS)**, **Shortest Job First (SJF)**, **Priority Scheduling**, and **Round Robin (RR)**. By allowing users to input process parameters such as arrival time, burst time, and priority, and then visualizing the scheduling sequence via **Gantt charts**, the tool offers a more intuitive and comprehensive understanding of how different strategies operate.

The simulation not only serves as a learning aid but also enables users to **analyze and compare** algorithm performance under various scenarios. This is especially useful for students, researchers, and educators involved in operating systems or real-time computing.

Through this project, we aim to create an accessible and interactive platform for understanding CPU scheduling mechanisms, while reinforcing theoretical concepts through hands-on experimentation and visual representation.

2.1 Background and Motivations

Process scheduling is a critical area of research and application in the field of computer science, particularly within the domain of operating systems. It deals with the selection of processes that are in the ready queue and allocating the CPU to one of them based on a specific algorithm. As modern computing systems have evolved to support multitasking, multiprocessing, and multi-user environments, the importance of efficient scheduling has increased significantly.

In today’s world, users expect their systems to be highly responsive and capable of handling multiple tasks simultaneously without noticeable delays. This demand has pushed operating systems to adopt sophisticated scheduling algorithms that can optimize CPU usage, reduce process waiting times, and ensure fairness and efficiency in task execution. Real-time systems, such as those used in robotics, telecommunications, and embedded systems, have even stricter requirements for timeliness and predictability, making scheduling a vital component of system design.

While scheduling algorithms like **First-Come, First-Served (FCFS)**, **Shortest Job First (SJF)**, **Priority Scheduling**, and **Round Robin (RR)** are extensively discussed in academic curricula, the understanding of these algorithms often remains abstract due to the lack of practical exposure. Students typically learn these concepts through static examples in textbooks or lectures, which do not adequately convey the dynamic behavior and trade-offs of different scheduling strategies when applied to real-world scenarios.

This gap between theory and practice serves as the main motivation for this project. The idea is to develop a **simulation-based process scheduler** that allows users to input various process parameters such as arrival time, burst time, and priority, and then observe how different scheduling algorithms handle those processes. By visualizing the order of execution, CPU allocation, and resulting performance metrics, users can gain a deeper understanding of the internal workings of each algorithm.

The project aims to serve as a **learning aid** that enhances conceptual clarity through interactive simulations and visual outputs like **Gantt charts** and performance tables. It helps users not only see how each algorithm works but also compare their relative efficiency under identical conditions. This approach provides a **hands-on learning experience**, enabling students and aspiring system developers to experiment with process scheduling in a controlled, educational environment.

Ultimately, the motivation behind this project is to make the abstract concepts of process scheduling more tangible, accessible, and engaging, thereby enriching the learning experience for those studying operating systems and real-time computing.

3.1 Problem Statement

In the field of operating systems, **CPU scheduling** is a foundational concept that determines how processes are selected for execution by the central processing unit. While this topic is covered extensively in theory, students often struggle to understand how different scheduling algorithms behave under various workloads and system conditions. Theoretical knowledge alone often proves insufficient in building a deep, intuitive understanding of process scheduling.

One of the primary challenges is that most academic courses on operating systems rely heavily on textbook examples and static illustrations to explain CPU scheduling algorithms such as **First-Come, First-Served (FCFS)**, **Shortest Job First (SJF)**, **Priority Scheduling**, and **Round Robin (RR)**. These examples do not reflect the dynamic nature of real-world process execution, nor do they provide the opportunity for learners to experiment with different scheduling techniques in an interactive environment.

As a result, students often face difficulties in:

- **Visualizing the scheduling order** and how the CPU is allocated to various processes over time.
- **Comparing the performance** of different algorithms using real-time metrics such as waiting time, turnaround time, and response time.
- **Understanding preemptive vs. non-preemptive behavior** in practical terms.
- **Observing how different process attributes** (e.g., arrival time, burst time, priority) influence scheduling decisions.

Moreover, there is a noticeable **lack of intuitive, educational tools or simulators** that allow users to input custom process data and immediately observe how various algorithms perform. Existing tools, if available, are often overly complex, outdated, or not tailored for academic use.

To address these gaps, this project aims to develop a **user-friendly and interactive process scheduling simulator** that:

- Accepts user-defined process data such as arrival time, burst time, and priority.
- Allows the selection of different scheduling algorithms.
- Provides visual feedback using **Gantt charts** to display process execution timelines.
- Calculates and displays key performance metrics such as **waiting time, turnaround time, and response time** for each process.
- Enables comparative analysis of algorithms under the same set of process conditions.

By providing a platform that bridges the gap between theory and practice, the project seeks to **enhance the learning experience** for students and help them build a clearer understanding of CPU scheduling. The tool is intended to be both educational and easy to use, making it suitable for classroom demonstrations, lab assignments, and self-study.

4.1 Objectives and Research Methodology

Objectives

The primary goal of this project is to simulate various CPU scheduling algorithms and provide users with a visual and analytical understanding of their behavior. In academic settings, theoretical learning often lacks a practical component that demonstrates how these algorithms function in real-time scenarios. Therefore, this project aims to bridge that gap by offering an interactive and educational experience.

The specific objectives of the project are as follows:

To implement a simulation of key CPU scheduling algorithms:The simulator will support major scheduling strategies such as **First-Come, First-Served (FCFS)**, **Shortest Job First**

(SJF), **Priority Scheduling**, and **Round Robin (RR)**. Both preemptive and non-preemptive versions will be considered where applicable.

1. **To provide an intuitive interface for user interface:**The application will allow users to easily input process details, including **process ID, arrival time, burst time, and priority**, through a simple graphical or command-line interface. The goal is to make the tool user-friendly and accessible to learners without requiring deep technical knowledge.
2. **To visualize process execution using Gantt charts** :A visual representation of the process execution order will be provided in the form of **Gantt charts**, allowing users to clearly observe how the CPU switches between processes over time based on the selected scheduling algorithm.
3. **To calculate and display essential performance metrics** :The simulator will compute important metrics such as **waiting time, turnaround time, and response time** for each process. These values are crucial for analyzing and comparing the efficiency of different scheduling strategies.
4. **To compare the performance of different algorithms under similar conditions**::The system will support side-by-side comparison of algorithms by using the same set of input processes. This will help users understand the trade-offs between different strategies in terms of fairness, efficiency, and responsiveness.
5. **To serve as an educational tool for learners and educators**:The project is designed to aid in the teaching and learning of operating system concepts, making it ideal for classroom use, lab sessions, and individual study.

Research Methodology

To achieve the objectives outlined above, a structured research and development methodology was followed. The methodology is divided into the following stages:

1. **Literature Review**
   A comprehensive study of **standard operating systems textbooks** and **research articles** was conducted to understand the theoretical foundations and practical implementations of CPU scheduling algorithms. Topics such as preemptive vs. non-preemptive scheduling, algorithm complexity, and performance metrics were explored in depth.
2. **Design**
   The design phase involved planning the architecture of the simulator, including the user interface layout, data structures, and flow of execution. **Flowcharts and pseudocode** were created for each scheduling algorithm to clearly define their logic and execution order. Design considerations also included modularity and scalability to allow future improvements or the addition of new algorithms.
3. **Implementation**
   The simulator was implemented using [**Insert programming language/platform here, e.g., Python with Tkinter/Java with Swing/C++**]. A modular approach was adopted to ensure that each component (input handling, algorithm logic, output visualization) could be developed and tested independently. Emphasis was placed on clean code structure and user-friendly interaction.

4. **Testing**
   Thorough testing was carried out using a variety of **sample inputs** and **edge cases** to ensure algorithm correctness and consistency. The outputs were validated against manual calculations and trusted academic examples. Special attention was given to the accuracy of performance metric calculations and the clarity of the Gantt chart displays.
5. **Analysis**
   After successful implementation and testing, the simulator was used to run **comparative experiments** on different scheduling algorithms. The performance of each algorithm was analyzed based on the calculated metrics. Graphs and tables were used to summarize and interpret the results, helping to highlight the strengths and weaknesses of each approach.
6. **Documentation and Feedback** : Throughout the project, progress was documented, and feedback was collected from peers and educators to refine the user interface, improve usability, and ensure the tool met its educational objectives

This structured methodology ensured that the development process was systematic, goal-oriented, and aligned with the educational purpose of the project. The resulting simulation tool is intended not only to demonstrate algorithm behavior but also to promote a deeper understanding of the underlying principles of CPU scheduling in operating systems.

5.1 Project Organization

To provide a structured and comprehensive overview of the development of the Process Scheduler project, this report is organized into six main chapters. Each chapter is designed to systematically present the key aspects of the project, from conceptual background to implementation details and final results. The organization of the report ensures clarity and continuity, making it easier for readers to follow the progression of the project.

The chapters are outlined as follows:

Chapter 2 Literature Review

This chapter presents a detailed review of the theoretical background relevant to process scheduling. It explores the concepts of CPU scheduling, the types of scheduling algorithms (such as FCFS, SJF, Priority Scheduling, and Round Robin), and their characteristics. In addition, it examines existing tools and simulators used for educational or industrial purposes, identifying gaps that this project aims to fill. The review helps to establish the foundation and rationale for developing the proposed simulator.

Chapter 3 System Design And Analysis

This chapter describes the functional and non-functional requirements of the system. It outlines the design process followed during development, including system architecture, data flow diagrams, and algorithm structures. The chapter also provides pseudocode or flowcharts for each scheduling algorithm implemented in the simulator. These designs guided the implementation phase and ensured consistency in the development process.

Chapter 4 Implementation

Here, the technical details of the project are discussed. It includes information on the programming language, libraries, and development tools used. The chapter describes the internal modules of the simulator, such as input handling, algorithm processing, and output visualization. The design and logic behind the graphical user interface (GUI) or command-line interface (CLI) are also covered, explaining how users interact with the system.

Chapter 5 Result and discussion

This section showcases the output generated by the simulator. It includes screenshots or diagrams of Gantt charts, tables displaying calculated performance metrics (such as waiting time, turnaround time, and response time), and graphical comparisons between algorithms. The discussion interprets these results, providing insights into the advantages and limitations of each scheduling method under different scenarios.

Chapter 6: Conclusion and Future Work

The final chapter summarizes the accomplishments of the project and reflects on its significance in the context of operating system education. It also discusses the limitations encountered during development and proposes potential improvements or extensions. Suggestions may include adding support for more advanced algorithms, integrating real-time scheduling policies, or enhancing the user interface for broader accessibility.

This chapter-wise organization ensures that the report comprehensively addresses all phases of the project—from theoretical grounding to practical application—while maintaining clarity and coherence throughout.

CHAPTER 2 PHASES OF SOFTWARE DEVELOPMENT CYCLE

1.1 Hardware Requirements

Component	Minimum Specification	Recommended Specification
Processor (CPU)	Dual-Core 2.0 GHz (e.g., Intel Core i3)	Quad-Core 2.5 GHz or higher (e.g., i5/i7)
RAM	4 GB	8 GB or more
Storage	500 MB available space	1 GB or more
Monitor	1024×768 resolution	1366×768 or higher
Input Devices	Standard Keyboard and Mouse	Standard Keyboard and Mouse

1.2 Software Requirements

Software Component	Description / Version
Operating System	Windows 10 / 11, Linux (Ubuntu 18.04+), or macOS
Java Development Kit	JDK 8 or later (e.g., JDK 17 recommended)
Integrated Development Environment (IDE)	Eclipse, IntelliJ IDEA, or NetBeans
Graphical UI Toolkit	Java Swing or JavaFX
Build Tool (Optional)	Apache Maven or Gradle
Version Control (Optional)	Git and GitHub
UML Tool (Optional)	StarUML, Lucidchart, or draw.io
Documentation Tools	Microsoft Word, Google Docs, or LaTeX

CHAPTER 3 CODING OF FUNCTIONS

1.1 .idea/libraries/CPUSchedulingSimulation_1_0_SNAPSHOT.xml

```
<component name="libraryTable">
<library name="CPUSchedulingSimulation-1.0-SNAPSHOT">
<CLASSES>
<root url="jar://$PROJECT_DIR$/target/CPUSchedulingSimulation-1.0-SNAPSHOT.jar!/"
/>
</CLASSES>
<JAVADOC />
<SOURCES />
</library>
</component>
```

misc.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
<component name="ProjectRootManager" version="2" languageLevel="JDK_15" project-
jdk-name="15" project-jdk-type="JavaSDK">
<output url="file://$PROJECT_DIR$/out" />
</component>
</project>
```

modules.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
<component name="ProjectRootManager" version="2" languageLevel="JDK_15" project-
jdk-name="15" project-jdk-type="JavaSDK">
<output url="file://$PROJECT_DIR$/out" />
</component>
</project>
```

workspace.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
<component name="ChangeListManager">
<list default="true" id="e39687cb-3a38-4bc1-8e6f-0ea714562383" name="Default
Changelist" comment="" />
<option name="SHOW_DIALOG" value="false" />
<option name="HIGHLIGHT_CONFLICTS" value="true" />
<option name="HIGHLIGHT_NON_ACTIVE_CHANGELIST" value="false" />
<option name="LAST_RESOLUTION" value="IGNORE" />
</component>
<component name="ProjectId" id="1tqLAqD6FmdM1jjVx4HtDH5dh6Z" />
<component name="ProjectViewState">
```

```
<option name="hideEmptyMiddlePackages" value="true" />
<option name="showLibraryContents" value="true" />
<option name="showMembers" value="true" />
</component>
<component name="PropertiesComponent">
<property name="RunOnceActivity.OpenProjectViewOnStart" value="true" />
<property name="last_opened_file_path" value="$PROJECT_DIR$" />
</component>
<component name="SpellCheckerSettings" RuntimeDictionaries="0" Folders="0"
CustomDictionaries="0" DefaultDictionary="application-level" UseSingleDictionary="true"
transferred="true" />
<component name="TaskManager">
<task active="true" id="Default" summary="Default task">
<changelist id="e39687cb-3a38-4bc1-8e6f-0ea714562383" name="Default Changelist"
comment="" />
<created>1623491532844</created>
<option name="number" value="Default" />
<option name="presentableId" value="Default" />
<updated>1623491532844</updated>
</task>
<servers />
</component>
</project>
```

Main

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.  See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */
  import java.awt.Dimension;
  import java.awt.Point;
```

```
/** An object that encapsulates position and (optionally) size for
* Absolute positioning of components.
*
* @see AbsoluteLayout
* @version 1.01, Aug 19, 1998
*/
public class AbsoluteConstraints implements java.io.Serializable {
 /** generated Serialized Version UID */
 static final long serialVersionUID = 5261460716622152494L;

 /** The X position of the component */
 public int x;
 /** The Y position of the component */
 public int y;
 /** The width of the component or -1 if the component's preferred width should be used */
 public int width = -1;
 /** The height of the component or -1 if the component's preferred height should be used */
 public int height = -1;

 /** Creates a new AbsoluteConstraints for specified position.
 * @param pos The position to be represented by this AbsoluteConstraints
 */
 public AbsoluteConstraints(Point pos) {
    this (pos.x, pos.y);
 }

 /** Creates a new AbsoluteConstraints for specified position.
 * @param x The X position to be represented by this AbsoluteConstraints
 * @param y The Y position to be represented by this AbsoluteConstraints
 */
 public AbsoluteConstraints(int x, int y) {
    this.x = x;
    this.y = y;
 }

 /** Creates a new AbsoluteConstraints for specified position and size.
 * @param pos  The position to be represented by this AbsoluteConstraints
 * @param size The size to be represented by this AbsoluteConstraints or null
 *            if the component's preferred size should be used
 */
 public AbsoluteConstraints(Point pos, Dimension size) {
 this.x = pos.x;
 this.y = pos.y;
 if (size != null) {
 this.width = size.width;
```

```
        this.height = size.height;
    }
}

/** Creates a new AbsoluteConstraints for specified position and size.
* @param x      The X position to be represented by this AbsoluteConstraints
* @param y      The Y position to be represented by this AbsoluteConstraints
* @param width  The width to be represented by this AbsoluteConstraints or -1 if the
*               component's preferred width should be used
* @param height The height to be represented by this AbsoluteConstraints or -1 if the
*               component's preferred height should be used
*/
public AbsoluteConstraints(int x, int y, int width, int height) {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
}
/** @return The X position represented by this AbsoluteConstraints */
public int getX () {
    return x;
}

/** @return The Y position represented by this AbsoluteConstraints */
public int getY () {
    return y;
}

/** @return The width represented by this AbsoluteConstraints or -1 if the
* component's preferred width should be used
*/
public int getWidth () {
    return width;
}
/** @return The height represented by this AbsoluteConstraints or -1 if the
* component's preferred height should be used
*/
public int getHeight () {
    return height;
}

public String toString () {
 return super.toString () +" [x="+x+", y="+y+", width="+width+", height="+height+"]";
}
}
import java.awt.*;
```

```
/** AbsoluteLayout is a LayoutManager that works as a replacement for "null" layout to
* allow placement of components in absolute positions.
*
* @see AbsoluteConstraints
* @version 1.01, Aug 19, 1998
*/
public class AbsoluteLayout implements LayoutManager2, java.io.Serializable {
/** generated Serialized Version UID */
static final long serialVersionUID = -1919857869177070440L;
/** Adds the specified component with the specified name to
* the layout.
* @param name the component name
* @param comp the component to be added
*/
 public void addLayoutComponent(String name, Component comp) {
 throw new IllegalArgumentException();
  }
 /** Removes the specified component from the layout.
 * @param comp the component to be removed
 */
 public void removeLayoutComponent(Component comp) {
 constraints.remove(comp);
 }

 /** Calculates the preferred dimension for the specified
* panel given the components in the specified parent container.
  * @param parent the component to be laid out
  *
  * @see #minimumLayoutSize
  */
  public Dimension preferredLayoutSize(Container parent) {
  int maxWidth = 0;
  int maxHeight = 0;
  for (java.util.Enumeration e = constraints.keys(); e.hasMoreElements();) {
  Component comp = (Component)e.nextElement();
  AbsoluteConstraints ac = (AbsoluteConstraints)constraints.get(comp);
  Dimension size = comp.getPreferredSize();
  int width = ac.getWidth ();
  if (width == -1) width = size.width;
  int height = ac.getHeight ();
  if (height == -1) height = size.height;
  if (ac.x + width > maxWidth)
  maxWidth = ac.x + width;
  if (ac.y + height > maxHeight)
  maxHeight = ac.y + height;
  }
```

```
return new Dimension (maxWidth, maxHeight);
}
/** Calculates the minimum dimension for the specified
* panel given the components in the specified parent container.
* @param parent the component to be laid out
* @see #preferredLayoutSize
*/
public Dimension minimumLayoutSize(Container parent) {
int maxWidth = 0;
int maxHeight = 0;
for (java.util.Enumeration e = constraints.keys(); e.hasMoreElements();) {
Component comp = (Component)e.nextElement();
AbsoluteConstraints ac = (AbsoluteConstraints)constraints.get(comp);
Dimension size = comp.getMinimumSize();
int width = ac.getWidth ();
if (width == -1) width = size.width;
int height = ac.getHeight ();
if (height == -1) height = size.height;
if (ac.x + width > maxWidth)
maxWidth = ac.x + width;
if (ac.y + height > maxHeight)
maxHeight = ac.y + height;
}
return new Dimension (maxWidth, maxHeight);
}
/** Lays out the container in the specified panel.
* @param parent the component which needs to be laid out
*/
 public void layoutContainer(Container parent) {
 for (java.util.Enumeration e = constraints.keys(); e.hasMoreElements();) {
 Component comp = (Component)e.nextElement();
 AbsoluteConstraints ac = (AbsoluteConstraints)constraints.get(comp);
 Dimension size = comp.getPreferredSize();
 int width = ac.getWidth ();
 if (width == -1) width = size.width;
 int height = ac.getHeight ();
 if (height == -1) height = size.height;
 comp.setBounds(ac.x, ac.y, width, height);
 }
 }
  /** Adds the specified component to the layout, using the specified
 * constraint object.
 * @param comp the component to be added
 * @param constr  where/how the component is added to the layout.
 */
  public void addLayoutComponent(Component comp, Object constr) {
```

```
if (!(constr instanceof AbsoluteConstraints))
throw new IllegalArgumentException();
constraints.put(comp, constr);
}
/** Returns the maximum size of this component.
* @see java.awt.Component#getMinimumSize()
* @see java.awt.Component#getPreferredSize()
* @see LayoutManager
*/
public Dimension maximumLayoutSize(Container target) {
return new Dimension(Integer.MAX_VALUE, Integer.MAX_VALUE);
}
/** Returns the alignment along the x axis.  This specifies how
* the component would like to be aligned relative to other
* components.  The value should be a number between 0 and 1
* where 0 represents alignment along the origin, 1 is aligned
* the furthest away from the origin, 0.5 is centered, etc.
*/
public float getLayoutAlignmentX(Container target) {
return 0;
}
/** Returns the alignment along the y axis.  This specifies how
* the component would like to be aligned relative to other
* components.  The value should be a number between 0 and 1
* where 0 represents alignment along the origin, 1 is aligned
* the furthest away from the origin, 0.5 is centered, etc.
*/
public float getLayoutAlignmentY(Container target) {
return 0;
}
** Invalidates the layout, indicating that if the layout manager
* has cached information it should be discarded.
 */
 public void invalidateLayout(Container target) {
 }
/** A mapping <Component, AbsoluteConstraints> */
protected java.util.Hashtable constraints = new java.util.Hashtable();
}
```

FCFS.java

```
import java.io.*;
Import javax.swing.JOptionPane;
public class FCFSPolicy implements Scheduler {
File sourceFile;
Node head, tail;
FCFSPolicy(File sourceFile) {
```

```
this.sourceFile = sourceFile;
head = tail = null;
}
public boolean isEmpty() { return head == null; }
public void enqueue(Job job) {
Node newNode = new Node(job);
if(isEmpty()) {
   head = tail = newNode;
} else {
   tail.next = newNode;
   tail = tail.next;
}
}
public Job dequeue() {
If(isEmpty()) {
 return null;
 } else {
 Job tempJob = head.job;
 head = head.next;
 tempJob.waitTime = System.nanoTime() - tempJob.startTime;
 return tempJob;
 }
 }
  public void allocateCPU(Job job) {
  try {
  BufferedReader bufReader = new BufferedReader(new FileReader(sourceFile))
 String str = new String();
 while((str = bufReader.readLine()) != null) {
 }String remProcesses = getRemainingProcesses();
 } catch(Exception ex)
  JOptionPane.showMessageDialog(null, "IO Error");
  Return}}
  public void enqueueAtHead(Job job)
  Node newNode = new Node(job);
   if(isEmpty()) head = newNode;
   else {
    Node temp = head;
    head = newNode;
    head.next = temp;
   }
   }
   public Node peek() {
   return head;
   }
   public String getRemainingProcesses() {
  Node trav = head;
```

```
String rem = new String();
 if(isEmpty()) return "";
 else {
 while(trav != null) {
 rem += trav.job.processID + " ";
 trav = trav.next;
 }
 }
  return rem;
 }
 }
```

Roundrobin.java

```
import java.io.*;
import javax.swing.*;

public class RoundRobinPolicy implements Scheduler {
static final long TIME_QUANTUM = 140; // Quantum of time is set to 140ns
File sourceFile;
Node head = null;
Node tail = null;
RoundRobinPolicy(File sourceFile) {
this.sourceFile = sourceFile;
}
 public void enqueue(Job job) {
 Node newNode = new Node(job);
 if (isEmpty()) {
 head = tail = newNode;
 } else {
  tail.next = newNode;
  tail = tail.next;
 }
 }
public void allocateCPU(Job job) {
try {
BufferedReader bufReader = new BufferedReader(new FileReader(sourceFile));
String str;
while ((str = bufReader.readLine()) != null) {
// Read and parse job data if needed
}

String remProcesses = getRemainingProcesses();
// Optional: Display or log remProcesses
} catch (Exception ex) {
JOptionPane.showMessageDialog(null, "IO Error");
return;
```

```
 }
 }
 public Job dequeue() {
 if (isEmpty())
 return null;
 } else {
  Job tempJob = head.job;
  head = head.next;
  tempJob.waitTime = System.nanoTime() - tempJob.startTime;
  return tempJob;
  }
 }
 public boolean isEmpty() {
 return head == null;
 }
 public String getRemainingProcesses() {
Node trav = head;
StringBuilder rem = new StringBuilder();
if (isEmpty()) {
 return "";
 } else {
 while (trav != null) {
 rem.append(trav.job.processID).append(" ");
 trav = trav.next;
 }
 }
 return rem.toString().trim();
 public Node peek() {
     return head;
  }
}
```

Job.java

```
import javax.swing.*;
public class Job {
long burstTime;
long arrivalTime;
String processID;
long startTime;
long endTime;
long waitTime;
JProgressBar progressBar;
JLabel burstTimeLabel;
int lastRemainingBurst;
JLabel waitTimeLabel;
```

```
 int pBarValue = 0;
 int priority;
 public Job(String processID, long arrivalTime, long burstTime, long startTime) {
 this.processID = processID;
 this.arrivalTime = arrivalTime;
 this.burstTime = burstTime;
 this.startTime = startTime;
 lastRemainingBurst = (int) burstTime;
}
}
```

Node.java

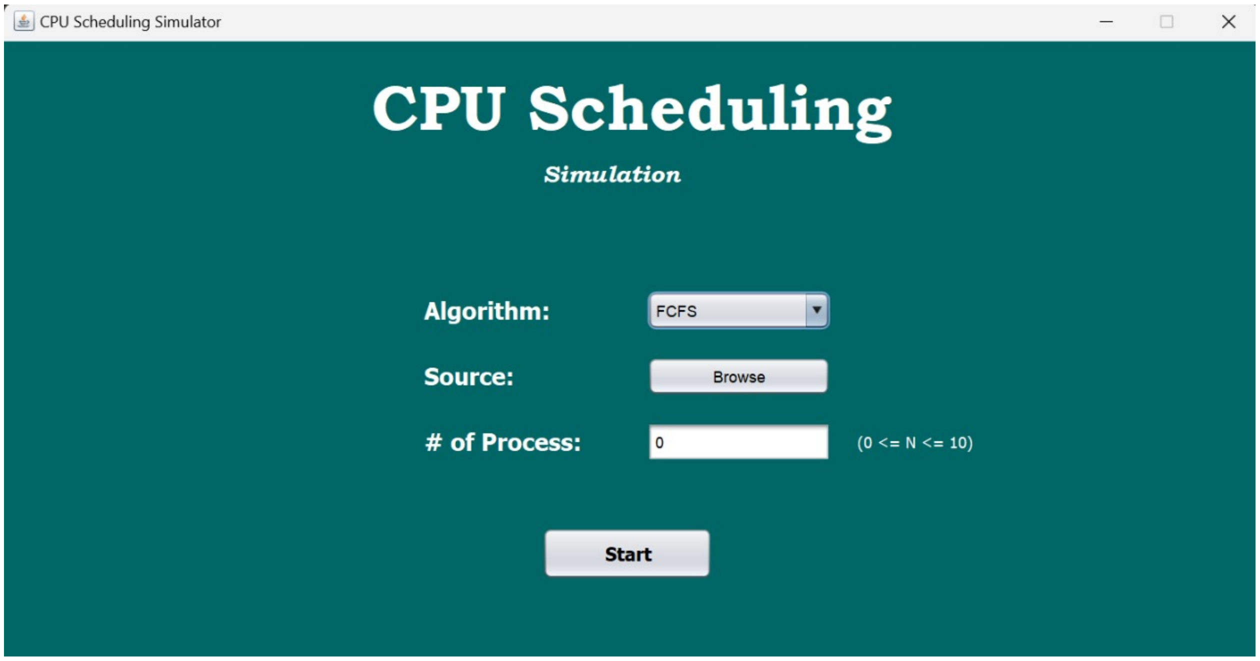
```
public class Node {
Node next;
Job job;
Node(Job job) {
this.job = job;
next = null;
}
}
```

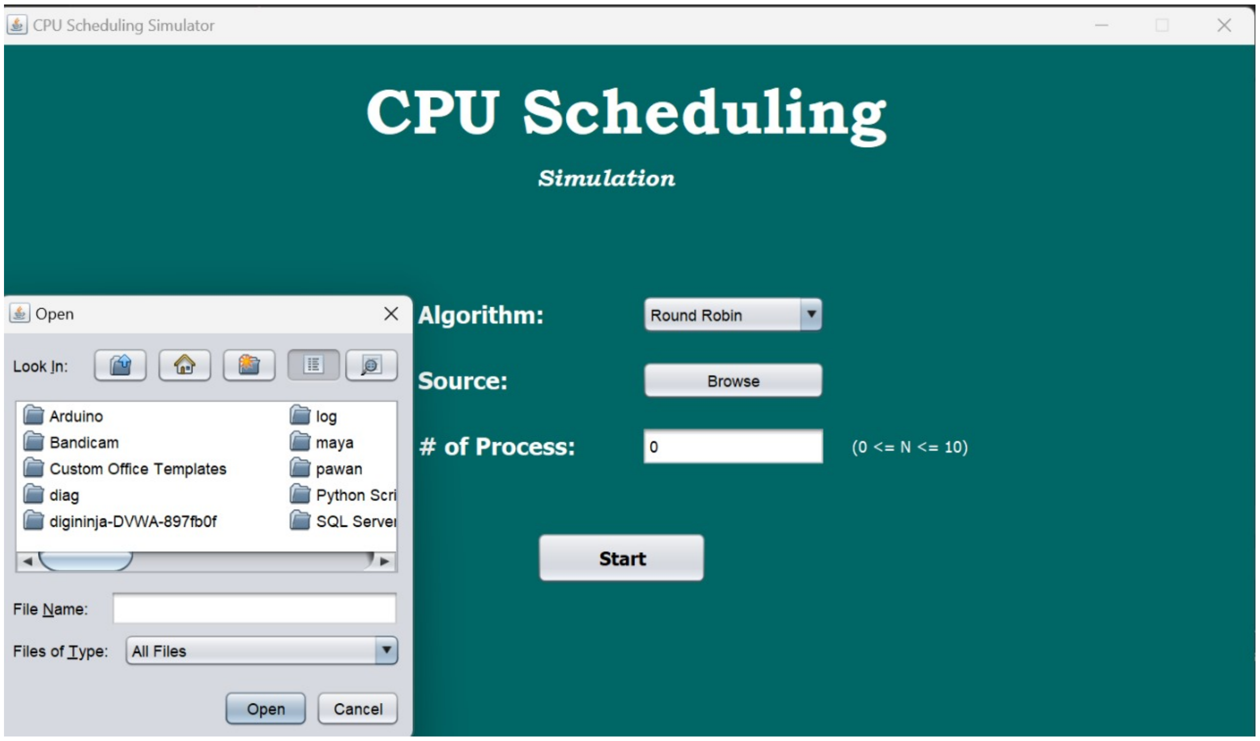
Prioritypolicy.java

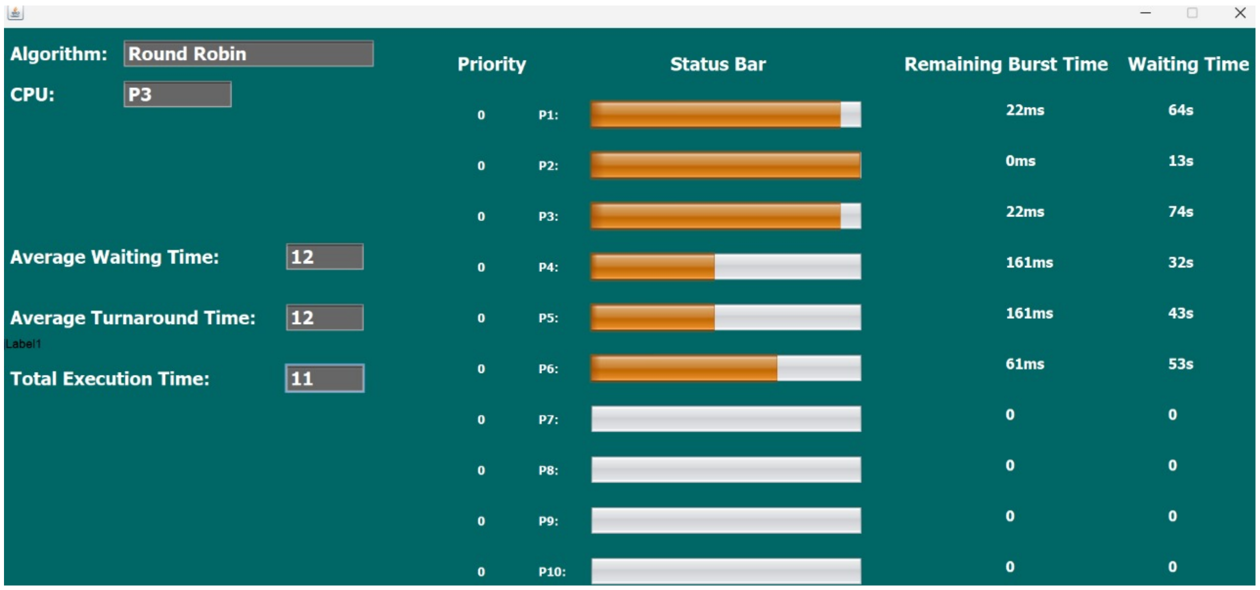
```
import java.io.*;
import javax.swing.JOptionPane;
public class PriorityPolicy implements Scheduler {
File sourceFile;
Node head, tail;
PriorityPolicy(File sourceFile) {
this.sourceFile = sourceFile;
head = tail = null;
}
public boolean isEmpty() { return head == null; }
public void enqueue(Job job) {
Node newNode = new Node(job);
if(isEmpty()) {
 head = tail = newNode;
 } else {
 tail.next = newNode;
 tail = tail.next;
 }
 }
  public Job dequeue() {
  if(isEmpty()) {
  return null;}
  else {
```

```
Job tempJob = head.job;
head = head.next;
tempJob.waitTime = System.nanoTime() - tempJob.startTime;
return tempJob;}    }
public void allocateCPU(Job job) {
try {
BufferedReader bufReader = new BufferedReader(new FileReader(sourceFile));
String str = new String();
while((str = bufReader.readLine()) != null) {
}
String remProcesses = getRemainingProcesses();
} catch(Exception ex) {
JOptionPane.showMessageDialog(null, "IO Error");
return;} }
public void enqueueAtHead(Job job) {
Node newNode = new Node(job);
if(isEmpty()) head = newNode;
else {
Node temp = head;
head = newNode;
head.next = temp;
}
}
 public Node peek() {
 return head;
 }
public String getRemainingProcesses() {
Node trav = head;
String rem = new String();
if(isEmpty()) return "";
else {
while(trav != null) {
 rem += trav.job.processID + " ";
 trav = trav.next;
 }
 }
 return rem;
 }
 }
```

CHAPTER 4 SNAPSHOTS







Algorithm: Round Robin

CPU: P3

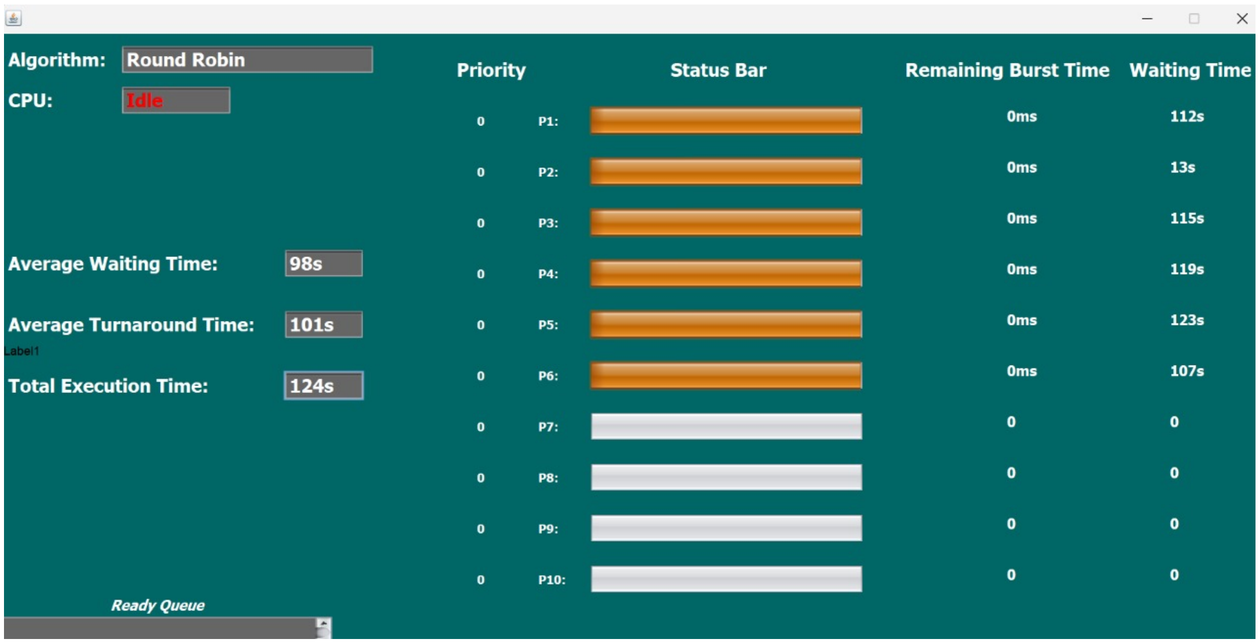
Average Waiting Time: 12

Average Turnaround Time: 12

Label1

Total Execution Time: 11

Priority		Status Bar	Remaining Burst Time	Waiting Time
0	P1:		22ms	64s
0	P2:		0ms	13s
0	P3:		22ms	74s
0	P4:		161ms	32s
0	P5:		161ms	43s
0	P6:		61ms	53s
0	P7:		0	0
0	P8:		0	0
0	P9:		0	0
0	P10:		0	0



Algorithm: Round Robin

CPU: Idle

Average Waiting Time: 98s

Average Turnaround Time: 101s

Label1

Total Execution Time: 124s

Ready Queue

Priority		Status Bar	Remaining Burst Time	Waiting Time
0	P1:		0ms	112s
0	P2:		0ms	13s
0	P3:		0ms	115s
0	P4:		0ms	119s
0	P5:		0ms	123s
0	P6:		0ms	107s
0	P7:		0	0
0	P8:		0	0
0	P9:		0	0
0	P10:		0	0

CHAPTER 5 LIMITATIONS

Although the Process Scheduler Project fulfills its intended functionality and accurately implements Round Robin and Priority Scheduling algorithms, certain limitations affect its scalability, efficiency, and real-world applicability. These limitations are outlined below.

Limited Scalability: The current implementation uses static arrays (e.g., in MaxPriorityQueue) with manual resizing, which can become inefficient as the number of jobs increases. While suitable for academic or small-scale simulation, it may struggle with handling hundreds or thousands of concurrent processes as seen in real operating systems.

Lack of Preemption Support: Although Round Robin inherently supports preemption via time quantum, the current simulation does not fully emulate interrupt-driven scheduling. The allocation is simulated based on static input rather than dynamically preempting based on real CPU or I/O events.

Incomplete Job I/O Simulation: The scheduler only focuses on CPU burst times and does not account for I/O bursts or blocked states, which are critical in real-world multiprogramming environments. This limits the simulation's accuracy for comprehensive system-level process handling.

Console and Basic GUI Only: While the GUI (if implemented using Java Swing or JavaFX) offers basic visual output or table views, it lacks advanced user interactivity and visualizations like Gantt charts or real-time CPU usage graphs. This makes it less suitable for users expecting a graphical simulation environment.

Single-Core Simulation: The scheduler operates under the assumption of a single CPU core, not taking into account multi-core or parallel processing environments. As modern systems are multi-threaded, this is a significant simplification.

Error Handling and Input Validation: Input handling is minimal and assumes valid job data. Improper input files or missing fields can cause runtime exceptions, as comprehensive validation or robust error recovery mechanisms have not been implemented.

No Persistent Data Logging: The system does not log process execution history or store results in external files (e.g., .csv, .txt). This restricts the ability to analyze scheduler performance post-simulation or use the output for research/benchmarking.

Despite these limitations, the project provides a solid foundation for understanding core CPU scheduling concepts. With further development — such as dynamic thread management, real-time input handling, and detailed UI — the project could be extended into a more robust process management simulator

CHAPTER 6 ENHANCEMENTS

While the current version of the Process Scheduler successfully implements core scheduling algorithms, it can be significantly improved in future iterations. The following are key areas where enhancements can be made.

Real-Time Job Execution Simulation

Instead of static input from files, future versions could allow users to add processes at runtime. This would simulate real-time environments more accurately, making it possible to demonstrate how schedulers handle dynamic workloads.

Support for Multi-Core Simulation

Introducing multi-core or parallel scheduling would align the simulator with modern computing systems. Implementing thread-based execution in Java would allow simulation of concurrent process execution across multiple CPU cores.

Integration of Additional Scheduling Algorithms

To broaden the educational scope, more scheduling algorithms like:Shortest Remaining Time First (SRTF),Multilevel Queue Scheduling,EDF can be incorporated with an option for users to select the scheduling method.

Enhanced Error Handling and Input Validation

Robust input validation (e.g., checking for missing fields or negative burst times) would improve the user experience and make the application more reliable, especially for larger test cases or bulk input data files.

Performance Metrics Dashboard

Implementing a results panel that displays:Average waiting time,Average turnaround time,CPU utilization,would provide detailed insights into the efficiency of each algorithm and allow easy comparison.

File Export and Reporting

Enable exporting of scheduling results (e.g., .csv or .txt) to allow users to keep logs, conduct analysis, or include results in academic reports and documentation.

Web-Based or Mobile Version

As an advanced enhancement, the application could be converted into a web-based or Android mobile app, enabling portability and wider accessibility to users.

CHAPTER 7 CONCLUSION

The **Process Scheduler Project** successfully demonstrates the implementation and functioning of core CPU scheduling algorithms, specifically Round Robin and Priority Scheduling, using the Java programming language. Through this project, key principles of process management in operating systems are simulated, offering valuable insights into how modern CPUs handle multiple tasks in a controlled and efficient manner.

This scheduler system accepts a set of jobs (processes), manages their execution order based on defined scheduling policies, and simulates CPU allocation. The use of a custom-built Max Priority Queue and a linked-list-based queue structure (for Round Robin) showcases the practical application of data structures in system-level programming.

The project emphasizes the importance of scheduling algorithms in achieving balanced CPU utilization, minimizing turnaround time, and improving overall system responsiveness. By simulating job arrival, execution, and queue management, it allows users to visualize the internal decision-making processes of an operating system's scheduler.

Additionally, the project reinforces object-oriented programming concepts, file handling in Java, and modular code structure—making it a strong educational tool for understanding operating system behavior and Java development.

While the current version fulfills its basic goals, it also highlights opportunities for future growth. Adding more features like real-time execution, graphical Gantt charts, and multi-core simulation would enhance the realism and user experience of the project.

This project has not only deepened our understanding of process scheduling but also emphasized the real-world application of algorithms and data structures in software development. It lays the foundation for further research and enhancement in the domain of operating system simulation.

This project also served as a practical demonstration of bridging theoretical knowledge with hands-on implementation. Concepts such as process prioritization, queue management, and execution timing—typically covered in classroom settings—were translated into working Java code. This experience has strengthened our analytical thinking and problem-solving skills while enhancing our proficiency in Java programming. By working on real-world inspired problems like CPU scheduling, we gained valuable exposure to how low-level operating system functions are modeled, tested, and optimized in software environments.

REFERENCES

.

[1] A. Silberschatz B., P. Galvin B., and G. Gagne, *Operating system concepts*, 10th Edition. Wiley.

[2] Linux Kernel Organization, Linux Kernel Documentation–CPU Scheduling. Available : https://www.kernel.org/doc/html/latest/

[3] ACM Digital Library, Scheduling Algorithms. Available :https://dl.acm.org/

[4] Scikit - learn Developers, Scikit-Learn – Machine Learning for Scheduling. Available: https://scikit-learn.org/stable/

[5] Open AI, Chat GPT Documentation. Available: https://openai.com/chatgpt

[6] Oracle, *Java Platform, Standard Edition – API Documentation*. Available: https://docs.oracle.com/javase/8/docs/api/