**PROCESS SCHEDULER SIMULATION**

A PROJECT REPORT

*Submitted by*

**SALONI BHARDWAJ [Reg No: RA2211003010268]   
SHOVIK BANERJEE [Reg No: RA2211003010270]**

**ARCHISMAN HES [Reg No: RA2211003010273]**

*Under the Guidance of*

**DR. M. ELIAZER**

Assistant Professor(Sr. Grade), Department of Computing Technology

*In partial fulfilment of the requirements for the degree of* **BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

****

**DEPARTMENT OF COMPUTING TECHNOLOGIES**

**COLLEGE OF ENGINEERING AND TECHNOLOGY SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR – 603 203**

**NOVEMBER 2023**



**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR – 603 203**

**BONAFIDE CERTIFICATE**

Certified that this B.Tech project report titled “**PROCESS SCHEDULER SIMULATION**” is the bonafide work of Saloni Bhardwaj [Reg. No.: RA2211003010268], Shovik Banerjee [Reg. No.RA2211003010270], Archisman Hes[Reg. No.RA2211003010273] and who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

**DR. M.ELIAZER**

Assistant Professor

Department of Computing technology

SRM Institute of Science and Technology

Kattankulathur

**SRM Institute of Science and Technology**

**Own Work Declaration Form**

**Degree/ Course :** B.Tech in Computer Science and Engineering

**Student Names :** Saloni Bhardwaj,Shovik Banerjee,Archisman Hes

**Registration Number:** RA2211003010268, RA2211003010270, RA2211003010273

**Title of Work :** Process Scheduler Simulation

We hereby certify that this assessment compiles with the University’s Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

• Clearly references / listed all sources as appropriate

• Referenced and put in inverted commas all quoted text (from books, web, etc.)

• Given the sources of all pictures, data etc. that are not my own

• Not made any use of the report(s) or essay(s) of any other student(s) either past or present

• Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources) • Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

|  |
| --- |
| **DECLARATION:** |
| I am aware of and understand the University’s policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above. |
| If you are working in a group, please write your registration numbers and sign with the date for every student in your group. |

v

**ACKNOWLEDGEMENT**

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr. T.V.Gopal**, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, Dr. **M. Pushpalatha ,**Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinator, **Dr. P. Rajashekhar** Assistant Professor, Panel Head, **Dr. M. Eliazer** , Associate Professor and members, **Dr. Rajkumar,** Assistant Professor, Dr. T. Karthick, Assistant professor and **Dr. S. Jeeva** , Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor**, Dr. Suresh Anand** ,Assistant Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr. A. Murugan**, Associate Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under his mentorship. He provided us with the freedom and support to explore the research topics of our interest. His passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank the Data Science and Business Systems Department staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

**SALONI BHARDWAJ[RA2211003010268]**

**SHOVIK BANERJEE[RA2211003010270]**

**ARCHISMAN HES[RA2211003010273]**

TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| ***S.No.*** | ***TITLE*** | ***PAGE NO.*** |
| ***1*** | ***Abstract*** | ***8*** |
| ***2*** | 1. ***Introduction***    1. ***General***    2. ***Purpose 2.3 Scope 2.4 Process Scheduler*** | ***10*** |
| ***3*** | ***Literature Review*** | ***13*** |
| ***4*** | ***Proposed methodology*** | ***16*** |
| ***5*** | ***Program Code*** | ***20*** |
| ***6*** | ***Output*** | ***24*** |
| ***6*** | ***Conclusion*** | ***38*** |
| ***7*** | ***Future Scope*** | ***39*** |
| ***8*** | ***References*** | ***40*** |
| ***9*** | ***Appendix*** | ***41*** |

**ABSTRACT**

A Process Scheduler Simulation serves as a vital instrument in comprehending the intricate mechanisms of how an operating system allocates CPU time to processes. Through its emulation, it unveils valuable insights into the behavior and performance of diverse scheduling algorithms employed in the management of processes. This comprehensive simulation involves several integral components. First and foremost, it requires the establishment of data structures that serve as the foundation for process management and scheduling. These data structures are instrumental in maintaining and organizing crucial information about processes, such as their priority, execution time, and current state. The heart of the simulation lies in the selection of a scheduling algorithm. Various algorithms, including First-Come-First-Serve (FCFS) and Shortest Job First (SJF), can be chosen based on the specific objectives and requirements of the system being studied. The selected algorithm guides the order in which processes are granted access to the CPU, significantly influencing system performance. Generating test cases is another essential aspect of the simulation. These test cases are designed to mimic real-world scenarios, allowing researchers to evaluate how the chosen scheduling algorithm performs under different conditions. They encompass a range of factors such as process arrival times, burst times, and priorities, enabling a comprehensive assessment. The implementation of the selected algorithm within the simulation framework is the next critical step. This involves coding the algorithm to accurately mimic its behavior in a real operating system. The simulation then runs these processes, simulating their execution on a virtual CPU. Finally, the simulation provides a platform for in-depth analysis of the results. Researchers can evaluate the system's performance by examining various metrics such as throughput, turnaround time, and waiting time. This analysis not only helps in assessing the effectiveness of the scheduling algorithm but also aids in understanding how resource utilization can be optimized. In summary, a Process Scheduler Simulation is an indispensable tool for studying the allocation of CPU time to processes. By defining data structures, selecting and implementing scheduling algorithms, generating test cases, and analyzing results, it contributes to a profound understanding of system performance and resource utilization optimization in various operating system environments..

**CHAPTER 1**

**INTRODUCTION**

**1.1 General**

A process scheduler is a fundamental component of modern operating systems, responsible for efficiently managing the execution of processes in a computer system. Its primary objective is to allocate and optimize the use of the CPU among multiple processes, ensuring optimal system performance and responsiveness in multitasking environments. The scheduler operates through various scheduling types, including long-term scheduling (job scheduling), medium-term scheduling, and short-term scheduling (CPU scheduling). Short-term scheduling involves selecting the next process from the ready queue and allocating CPU time. Several algorithms, such as First-Come-First-Served, Shortest Job Next, Round Robin, and Priority Scheduling, are employed for short-term scheduling, each with its own trade-offs. Priority-based scheduling allows for the execution of higher-priority processes first, but careful consideration is needed to prevent the potential issue of starvation for lower-priority processes. Context switching, the process of saving and loading process states during a switch, incurs overhead and must be minimized for efficiency. Real-time scheduling is crucial for meeting deadlines in time-sensitive systems, while interactive and batch systems prioritize response times and throughput, respectively. Dynamic scheduling, adjusting priorities or time quantum during runtime, is a feature in some modern operating systems. The careful design and implementation of a process scheduler are essential for achieving a balance between system responsiveness, throughput, and fairness in resource allocation, tailored to the specific goals and characteristics of the operating system and its supported applications.

**1.2 Purpose**

Through this project, we aim to deliver a feasible, and viable method to understand Process scheduler simulation plays a pivotal role in computer science and operating system development by providing a controlled environment for evaluating, comparing, and refining process scheduling algorithms. Through simulations, developers and researchers can assess the performance of different scheduling strategies under diverse workloads, aiding in algorithm selection and parameter tuning. The ability to analyze scalability, predict behavior, and study fault tolerance in simulated scenarios is invaluable for designing robust systems capable of handling various conditions. Additionally, process scheduler simulations serve as educational tools, offering insights into the inner workings of scheduling algorithms and facilitating hands-on learning experiences. These simulations also support the development and prototyping of new algorithms, allowing for iterative refinement before implementation in real operating systems. Ultimately, process scheduler simulations contribute to benchmarking efforts, establishing performance metrics and providing a foundation for advancing the efficiency and effectiveness of process scheduling in operating systems.

**1.3 Scope**

The scope of process scheduler simulation is broad and encompasses various facets crucial to the understanding, optimization, and advancement of process scheduling in operating systems. Through simulation, researchers and developers gain a comprehensive platform to evaluate the performance of diverse scheduling algorithms under different conditions, facilitating the identification of strengths, weaknesses, and trade-offs associated with each strategy. The scope extends to the analysis of scalability, enabling the examination of how scheduling algorithms perform as system loads and the number of processes increase. Additionally, simulations allow for the exploration of fault tolerance mechanisms, providing insights into how well schedulers adapt to unexpected events or errors. The educational value of process scheduler simulation is significant, offering students and practitioners an interactive environment to comprehend the intricacies of scheduling algorithms and their impact on system behavior. Furthermore, the scope includes the development and prototyping of new algorithms, allowing for iterative testing and refinement before real-world implementation. As an indispensable tool, process scheduler simulation serves to benchmark different strategies, establishing baseline metrics and contributing to ongoing efforts aimed at enhancing the efficiency and responsiveness of process scheduling in operating systems.

**1.4 Process Scheduler**

"Inefficient process scheduling within an operating system is a critical issue that hampers system performance and resource utilization. In today's dynamic computing environments, diverse workloads with varying priorities and execution times must be managed effectively. However, existing scheduling algorithms may not adequately address these challenges. The problem at hand is to design, implement, and evaluate a novel process scheduler that optimizes CPU time allocation for processes, considering factors like real-time constraints, multitasking, and resource contention. The scheduler should aim to minimize turnaround time, waiting time, and CPU idleness, ultimately enhancing system responsiveness and throughput. This problem requires a thorough investigation of existing scheduling algorithms and the development of innovative strategies to address the complex scheduling requirements of modern computing systems. The solution should balance the conflicting goals of fairness, efficiency, and responsiveness, while ensuring that high-priority and real-time tasks are prioritized. Additionally, the proposed scheduler should be evaluated through comprehensive simulations and real-world testing to ensure its effectiveness in various computing scenarios. The outcome of this research will lead to improved system performance, resource utilization, and user satisfaction in diverse computing environments."

**CHAPTER 2**

**LITERATURE REVIEW**

As of my last knowledge update in January 2022, I can't provide specific or up-to-date literature reviews on process scheduler simulation. However, I can offer a general overview of the types of research areas and key themes that are often explored in the literature regarding process scheduler simulation.

1. Scheduling Algorithms: Numerous studies focus on evaluating and comparing the performance of different scheduling algorithms through simulation. This includes classic algorithms like First-Come-First-Served (FCFS), Round Robin, Shortest Job Next (SJN), and more recent variations or hybrid approaches.

2. Real-Time Systems: Literature often delves into the simulation of process schedulers in real-time systems, where meeting deadlines is critical. This involves exploring scheduling algorithms designed to handle time-sensitive tasks and assessing their effectiveness in simulation environments.

3. Performance Metrics:Researchers commonly define and employ various performance metrics to evaluate the effectiveness of process scheduling algorithms. These metrics may include CPU utilization, throughput, turnaround time, waiting time, and response time.

4. Multicore Systems: With the prevalence of multicore processors, literature on process scheduler simulation may address how scheduling algorithms perform in these environments. This includes considerations for load balancing and resource allocation across multiple cores.

5. Dynamic Scheduling: Dynamic scheduling, where the scheduler adapts to changing conditions, is often explored in the literature. This involves studying how scheduling algorithms can dynamically adjust priorities or other parameters during runtime to optimize system performance.

6. Energy-Efficient Scheduling: Some studies may focus on simulating process schedulers with an emphasis on energy efficiency. This includes exploring algorithms and techniques to minimize power consumption while maintaining acceptable system performance.

7. Machine Learning and AI-Based Scheduling: Recent literature may touch upon the integration of machine learning or artificial intelligence techniques into process scheduling. Simulations might be used to evaluate the performance of these intelligent scheduling approaches.

8. Simulation Tools and Frameworks: Some literature reviews may discuss the simulation tools and frameworks commonly used for studying process schedulers. This could include discrete event simulation tools, performance modeling languages, or specialized simulation platforms.

9. Scalability and Robustness: Scalability testing is a common theme, exploring how scheduling algorithms perform as the system scales in terms of the number of processes, system load, or hardware resources. Additionally, research might investigate the robustness of scheduling algorithms under various conditions and potential failure scenarios.

10. Educational Simulations: Literature may cover the use of process scheduler simulations as educational tools, discussing how simulations contribute to teaching and understanding scheduling concepts in academic settings.

When conducting a literature review, it's essential to consider recent publications, conference proceedings, and journals in the field of operating systems, computer science, and simulation. Search databases such as IEEE Xplore, ACM Digital Library, and Google Scholar for the most up-to-date research articles on process scheduler simulation.

Additionally, recent literature explores the dynamic nature of scheduling algorithms, investigating their ability to adapt to changing conditions during runtime. Some studies focus on energy-efficient scheduling, aiming to minimize power consumption while maintaining satisfactory system performance. The integration of machine learning and artificial intelligence techniques into process scheduling has become a noteworthy trend, with researchers employing simulations to assess the performance of intelligent scheduling approaches. Scalability remains a critical consideration, with investigations into how scheduling algorithms perform as the system scales in terms of the number of processes, system load, and hardware resources. The robustness of scheduling algorithms under various conditions and potential failure scenarios is also a topic of interest, emphasizing the need for reliable scheduling mechanisms in real-world scenarios. Simulations, beyond their research utility, are recognized as effective tools for educational purposes, providing hands-on experiences and enhancing the understanding of scheduling concepts in academic settings. Literature also discusses the selection and application of simulation tools and frameworks, contributing to the methodological aspects of process scheduler simulation research. Overall, the literature reflects a comprehensive exploration of process scheduler simulation, addressing diverse aspects that contribute to the optimization and advancement of scheduling algorithms in operating systems.

**CHAPTER 3**

PROPOSED METHODOLOGY

**SCHEDULING AND IT’S TYPES**

Scheduling, in the context of computer science and operating systems, refers to the process of determining the order in which tasks, processes, or jobs are executed by a computer or other computing systems. It involves the allocation of system resources, primarily the central processing unit (CPU), to various tasks to ensure efficient utilization and effective management of these resources. Scheduling algorithms aim to prioritize and execute tasks in a manner that optimizes factors such as system responsiveness, throughput, fairness, and resource utilization. Different scheduling algorithms are employed based on the specific goals and constraints of the computing system, and they determine which task gets access to the CPU and for how long. Scheduling plays a vital role in ensuring the effective operation of multitasking, multi-user systems and is crucial in real-time computing applications, where meeting specific timing constraints is essential.

1. **First-Come-First-Serve (FCFS) scheduling** is one of the simplest and most straightforward scheduling algorithms used in operating systems. In FCFS scheduling, processes are executed in the order they arrive in the ready queue, with the first process to arrive being the first one to receive the CPU and execute.

Key characteristics of FCFS scheduling:

* Non-preemptive: FCFS is a non-preemptive scheduling algorithm, which means that once a process starts executing, it continues until it finishes or voluntarily releases the CPU.
* Simple to implement: FCFS is easy to understand and implement, making it suitable for simple systems.
* Lack of priority consideration: FCFS does not consider the priority of processes or their execution times. It simply follows the order in which processes arrive.
* Can lead to the "convoy effect": FCFS can result in the convoy effect, where a long process holding the CPU can cause shorter processes to wait, leading to inefficient resource utilization.
* Poor for interactive systems: FCFS is not well-suited for interactive or real-time systems where responsiveness is critical because it may result in long response times

1. **Shortest Job First (SJF) scheduling** is a CPU scheduling algorithm used in operating systems. It aims to minimize the average waiting time of processes by giving preference to the shortest job (process) in the ready queue. SJF is also known as Shortest Job Next (SJN) or Shortest Job First-Come-First-Served (SJF-FCFS).

Key characteristics of SJF scheduling:

* Preemptive and non-preemptive: SJF can be implemented in both preemptive and non-preemptive versions. In the preemptive version, a new job can interrupt the execution of the currently running job if a shorter job becomes available. In the non-preemptive version, the running job is not interrupted until it completes its execution.
* Dynamic priority: SJF essentially assigns priorities based on the expected burst time (time needed for a process to complete). The shorter the burst time, the higher the priority.
* Minimizes waiting time: SJF scheduling is designed to minimize the waiting time, which is the time a process spends in the ready queue before it gets CPU time.
* Potential for starvation: While SJF scheduling is efficient in terms of minimizing waiting times, it can lead to the starvation of longer jobs if a continuous stream of short jobs keeps arriving.

1. **Priority Scheduling - Non-Preemptive (PS-NP)** is a CPU scheduling algorithm used in operating systems. It assigns a priority value to each process, and the process with the highest priority is selected to run first. Unlike the preemptive version of priority scheduling, in non-preemptive priority scheduling, once a process is given the CPU, it continues to execute until it completes or voluntarily releases the CPU.

Key characteristics of Priority Scheduling - Non-Preemptive:

* Priority assignment: Each process is assigned a priority value based on its characteristics, such as its importance or resource requirements. Higher priority values indicate higher priority.
* Process selection: The process with the highest priority in the ready queue is selected to run. If multiple processes share the highest priority, the scheduling algorithm may use additional criteria, like First-Come-First-Serve (FCFS) within the same priority group.
* Lack of time-sharing: In non-preemptive priority scheduling, a process runs to completion without interruption. Only when it completes or enters the waiting state does the CPU become available for other processes with higher priorities.
* Potential for starvation: Lower-priority processes may experience starvation if higher-priority processes continually arrive in the system. To mitigate this, aging mechanisms can be implemented to increase the priority of waiting processes over time.
* Real-time systems: Priority scheduling is commonly used in real-time operating systems where certain tasks must meet strict timing constraints, and tasks are assigned priorities based on their deadlines and criticality.

1. **Priority Scheduling - Preemptive (PS-P)** is a CPU scheduling algorithm used in operating systems. It assigns a priority value to each process, and the process with the highest priority that is ready to run is given the CPU. However, unlike non-preemptive priority scheduling, in PS-P, a process currently executing can be preempted if a higher-priority process becomes available.

Key characteristics of Priority Scheduling - Preemptive:

* + Priority assignment: Each process is assigned a priority value based on various factors, such as the importance of the task, deadline constraints, or resource requirements. Higher priority values represent higher priority.
  + Process selection: The process with the highest priority in the ready queue is allowed to run on the CPU. If a higher-priority process arrives or becomes ready to run, the currently executing process can be preempted and moved back to the ready queue.
  + Time-sharing: Preemptive priority scheduling allows the CPU to be shared among processes more fairly and responsively. High-priority tasks can start promptly, and lower-priority tasks can still be executed.
* Avoiding starvation: Preemptive priority scheduling can help prevent starvation of lower-priority processes since they are given the opportunity to execute when the CPU becomes available.
* Real-time systems: PS-P is often used in real-time operating systems, particularly when tasks need to meet strict deadlines and the system must ensure that high-priority tasks are executed without delay.

1. **Round Robin Scheduling (RR)** is a widely used CPU scheduling algorithm in operating systems. It is designed to provide fair access to the CPU and ensure that no process monopolizes it for an extended period. In Round Robin scheduling, each process is assigned a fixed time quantum, and processes take turns executing for their allocated time, moving in a circular queue.

Key characteristics of Round Robin Scheduling:

* + Time quantum: The most defining feature of Round Robin scheduling is the time quantum or time slice, which is a fixed amount of time allocated to each process. When a process's turn arrives, it is allowed to run for the time quantum or until it voluntarily yields the CPU (e.g., due to I/O operations).
  + Preemptive: Round Robin is a preemptive scheduling algorithm, which means that if a process does not complete its execution within the time quantum, it is temporarily removed from the CPU, and the next process in the queue is given a chance to execute. The interrupted process is placed back at the end of the queue.
  + Fairness: Round Robin aims to provide fair access to the CPU for all processes. Because of the fixed time quantum, no process can monopolize the CPU for an extended period, ensuring that all processes get a chance to execute.
  + Response time: Round Robin provides good response times for interactive processes since they get a chance to execute frequently. However, it may not be as efficient as other scheduling algorithms for long-running CPU-bound tasks.

**CHAPTER 4**

**PROGRAM CODE**

* 1. **GUI :**

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Font;

import java.awt.Graphics;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.List;

import javax.swing.JButton;

import javax.swing.JComboBox;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.table.DefaultTableModel;

public class GUI

{

private JFrame frame;

private JPanel mainPanel;

private CustomPanel chartPanel;

private JScrollPane tablePane;

private JScrollPane chartPane;

private JTable table;

private JButton addBtn;

private JButton removeBtn;

private JButton computeBtn;

private JLabel wtLabel;

private JLabel wtResultLabel;

private JLabel tatLabel;

private JLabel tatResultLabel;

private JComboBox option;

private DefaultTableModel model;

public GUI()

{

model = new DefaultTableModel(new String[]{"Process", "AT", "BT", "Priority", "WT", "TAT"}, 0);

table = new JTable(model);

table.setFillsViewportHeight(true);

tablePane = new JScrollPane(table);

tablePane.setBounds(25, 25, 450, 250);

addBtn = new JButton("Add");

addBtn.setBounds(300, 280, 85, 25);

addBtn.setFont(new Font("Segoe UI", Font.PLAIN, 11));

addBtn.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

model.addRow(new String[]{"", "", "", "", "", ""});

}

});

removeBtn = new JButton("Remove");

removeBtn.setBounds(390, 280, 85, 25);

removeBtn.setFont(new Font("Segoe UI", Font.PLAIN, 11));

removeBtn.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

int row = table.getSelectedRow();

if (row > -1) {

model.removeRow(row);

}

}

});

chartPanel = new CustomPanel();

// chartPanel.setPreferredSize(new Dimension(700, 10));

chartPanel.setBackground(Color.WHITE);

chartPane = new JScrollPane(chartPanel);

chartPane.setBounds(25, 310, 450, 100);

wtLabel = new JLabel("Average Waiting Time:");

wtLabel.setBounds(25, 425, 180, 25);

tatLabel = new JLabel("Average Turn Around Time:");

tatLabel.setBounds(25, 450, 180, 25);

wtResultLabel = new JLabel();

wtResultLabel.setBounds(215, 425, 180, 25);

tatResultLabel = new JLabel();

tatResultLabel.setBounds(215, 450, 180, 25);

option = new JComboBox(new String[]{"FCFS", "SJF", "PS-NP", "PS-P", "RR"});

option.setBounds(390, 420, 85, 20);

computeBtn = new JButton("Compute");

computeBtn.setBounds(390, 450, 85, 25);

computeBtn.setFont(new Font("Segoe UI", Font.PLAIN, 11));

computeBtn.addActionListener(new ActionListener(){

@Override

public void actionPerformed(ActionEvent e) {

String selected = (String) option.getSelectedItem();

CPUScheduler scheduler;

switch (selected) {

case "FCFS":

scheduler = new FirstComeFirstServe();

break;

case "SJF":

scheduler = new ShortestJobFirst();

break;

case "PS-NP":

scheduler = new PriorityNonPreemptive();

break;

case "PS-P":

scheduler = new PriorityPreemptive();

break;

case "RR":

String tq = JOptionPane.showInputDialog("Time Quantum");

if (tq == null) {

return;

}

scheduler = new RoundRobin();

scheduler.setTimeQuantum(Integer.parseInt(tq));

break;

default:

return;

}

for (int i = 0; i < model.getRowCount(); i++)

{

String process = (String) model.getValueAt(i, 0);

int at = Integer.parseInt((String) model.getValueAt(i, 1));

int bt = Integer.parseInt((String) model.getValueAt(i, 2));

int pl;

if (selected.equals("PS-NP") || selected.equals("PS-P"))

{

if (!model.getValueAt(i, 3).equals(""))

{

pl = Integer.parseInt((String) model.getValueAt(i, 3));

}

else

{

pl = 1;

}

}

else

{

pl = 1;

}

scheduler.add(new Row(process, at, bt, pl));

}

scheduler.process();

for (int i = 0; i < model.getRowCount(); i++)

{

String process = (String) model.getValueAt(i, 0);

Row row = scheduler.getRow(process);

model.setValueAt(row.getWaitingTime(), i, 4);

model.setValueAt(row.getTurnaroundTime(), i, 5);

}

wtResultLabel.setText(Double.toString(scheduler.getAverageWaitingTime()));

tatResultLabel.setText(Double.toString(scheduler.getAverageTurnAroundTime()));

chartPanel.setTimeline(scheduler.getTimeline());

}

});

mainPanel = new JPanel(null);

mainPanel.setPreferredSize(new Dimension(500, 500));

mainPanel.add(tablePane);

mainPanel.add(addBtn);

mainPanel.add(removeBtn);

mainPanel.add(chartPane);

mainPanel.add(wtLabel);

mainPanel.add(tatLabel);

mainPanel.add(wtResultLabel);

mainPanel.add(tatResultLabel);

mainPanel.add(option);

mainPanel.add(computeBtn);

frame = new JFrame("CPU Scheduler Simulator");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setVisible(true);

frame.setResizable(false);

frame.add(mainPanel);

frame.pack();

}

public static void main(String[] args)

{

new GUI();

}

class CustomPanel extends JPanel

{

private List<Event> timeline;

@Override

protected void paintComponent(Graphics g)

{

super.paintComponent(g);

if (timeline != null)

{

// int width = 30;

for (int i = 0; i < timeline.size(); i++)

{

Event event = timeline.get(i);

int x = 30 \* (i + 1);

int y = 20;

g.drawRect(x, y, 30, 30);

g.setFont(new Font("Segoe UI", Font.BOLD, 13));

g.drawString(event.getProcessName(), x + 10, y + 20);

g.setFont(new Font("Segoe UI", Font.PLAIN, 11));

g.drawString(Integer.toString(event.getStartTime()), x - 5, y + 45);

if (i == timeline.size() - 1)

{

g.drawString(Integer.toString(event.getFinishTime()), x + 27, y + 45);

}

// width += 30;

}

// this.setPreferredSize(new Dimension(width, 75));

}

}

public void setTimeline(List<Event> timeline)

{

this.timeline = timeline;

repaint();

}

}

}

* 1. **FCFS Scheduling:**

import java.util.Collections;

import java.util.List;

public class FirstComeFirstServe extends CPUScheduler

{

@Override

public void process()

{

Collections.sort(this.getRows(), (Object o1, Object o2) -> {

if (((Row) o1).getArrivalTime() == ((Row) o2).getArrivalTime())

{

return 0;

}

else if (((Row) o1).getArrivalTime() < ((Row) o2).getArrivalTime())

{

return -1;

}

else

{

return 1;

}

});

List<Event> timeline = this.getTimeline();

for (Row row : this.getRows())

{

if (timeline.isEmpty())

{

timeline.add(new Event(row.getProcessName(), row.getArrivalTime(), row.getArrivalTime() + row.getBurstTime()));

}

else

{

Event event = timeline.get(timeline.size() - 1);

timeline.add(new Event(row.getProcessName(), event.getFinishTime(), event.getFinishTime() + row.getBurstTime()));

}

}

for (Row row : this.getRows())

{

row.setWaitingTime(this.getEvent(row).getStartTime() - row.getArrivalTime());

row.setTurnaroundTime(row.getWaitingTime() + row.getBurstTime());

}

}

}

* 1. **Priority Non Pre-Emptive:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class PriorityNonPreemptive extends CPUScheduler

{

@Override

public void process()

{

Collections.sort(this.getRows(), (Object o1, Object o2) -> {

if (((Row) o1).getArrivalTime() == ((Row) o2).getArrivalTime())

{

return 0;

}

else if (((Row) o1).getArrivalTime() < ((Row) o2).getArrivalTime())

{

return -1;

}

else

{

return 1;

}

});

List<Row> rows = Utility.deepCopy(this.getRows());

int time = rows.get(0).getArrivalTime();

while (!rows.isEmpty())

{

List<Row> availableRows = new ArrayList();

for (Row row : rows)

{

if (row.getArrivalTime() <= time)

{

availableRows.add(row);

}

}

Collections.sort(availableRows, (Object o1, Object o2) -> {

if (((Row) o1).getPriorityLevel()== ((Row) o2).getPriorityLevel())

{

return 0;

}

else if (((Row) o1).getPriorityLevel() < ((Row) o2).getPriorityLevel())

{

return -1;

}

else

{

return 1;

}

});

Row row = availableRows.get(0);

this.getTimeline().add(new Event(row.getProcessName(), time, time + row.getBurstTime()));

time += row.getBurstTime();

for (int i = 0; i < rows.size(); i++)

{

if (rows.get(i).getProcessName().equals(row.getProcessName()))

{

rows.remove(i);

break;

}

}

}

for (Row row : this.getRows())

{

row.setWaitingTime(this.getEvent(row).getStartTime() - row.getArrivalTime());

row.setTurnaroundTime(row.getWaitingTime() + row.getBurstTime());

}

}

}

* 1. **Priority Preemptive:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

public class PriorityPreemptive extends CPUScheduler

{

@Override

public void process()

{

Collections.sort(this.getRows(), (Object o1, Object o2) -> {

if (((Row) o1).getArrivalTime() == ((Row) o2).getArrivalTime())

{

return 0;

}

else if (((Row) o1).getArrivalTime() < ((Row) o2).getArrivalTime())

{

return -1;

}

else

{

return 1;

}

});

List<Row> rows = Utility.deepCopy(this.getRows());

int time = rows.get(0).getArrivalTime();

while (!rows.isEmpty())

{

List<Row> availableRows = new ArrayList();

for (Row row : rows)

{

if (row.getArrivalTime() <= time)

{

availableRows.add(row);

}

}

Collections.sort(availableRows, (Object o1, Object o2) -> {

if (((Row) o1).getPriorityLevel()== ((Row) o2).getPriorityLevel())

{

return 0;

}

else if (((Row) o1).getPriorityLevel() < ((Row) o2).getPriorityLevel())

{

return -1;

}

else

{

return 1;

}

});

Row row = availableRows.get(0);

this.getTimeline().add(new Event(row.getProcessName(), time, ++time));

row.setBurstTime(row.getBurstTime() - 1);

if (row.getBurstTime() == 0)

{

for (int i = 0; i < rows.size(); i++)

{

if (rows.get(i).getProcessName().equals(row.getProcessName()))

{

rows.remove(i);

break;

}

}

}

}

for (int i = this.getTimeline().size() - 1; i > 0; i--)

{

List<Event> timeline = this.getTimeline();

if (timeline.get(i - 1).getProcessName().equals(timeline.get(i).getProcessName()))

{

timeline.get(i - 1).setFinishTime(timeline.get(i).getFinishTime());

timeline.remove(i);

}

}

Map map = new HashMap();

for (Row row : this.getRows())

{

map.clear();

for (Event event : this.getTimeline())

{

if (event.getProcessName().equals(row.getProcessName()))

{

if (map.containsKey(event.getProcessName()))

{

int w = event.getStartTime() - (int) map.get(event.getProcessName());

row.setWaitingTime(row.getWaitingTime() + w);

}

else

{

row.setWaitingTime(event.getStartTime() - row.getArrivalTime());

}

map.put(event.getProcessName(), event.getFinishTime());

}

}

row.setTurnaroundTime(row.getWaitingTime() + row.getBurstTime());

}

}

}

* 1. **Round Robin Scheduling:**

import java.util.Collections;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

public class RoundRobin extends CPUScheduler

{

@Override

public void process()

{

Collections.sort(this.getRows(), (Object o1, Object o2) -> {

if (((Row) o1).getArrivalTime() == ((Row) o2).getArrivalTime())

{

return 0;

}

else if (((Row) o1).getArrivalTime() < ((Row) o2).getArrivalTime())

{

return -1;

}

else

{

return 1;

}

});

List<Row> rows = Utility.deepCopy(this.getRows());

int time = rows.get(0).getArrivalTime();

int timeQuantum = this.getTimeQuantum();

while (!rows.isEmpty())

{

Row row = rows.get(0);

int bt = (row.getBurstTime() < timeQuantum ? row.getBurstTime() : timeQuantum);

this.getTimeline().add(new Event(row.getProcessName(), time, time + bt));

time += bt;

rows.remove(0);

if (row.getBurstTime() > timeQuantum)

{

row.setBurstTime(row.getBurstTime() - timeQuantum);

for (int i = 0; i < rows.size(); i++)

{

if (rows.get(i).getArrivalTime() > time)

{

rows.add(i, row);

break;

}

else if (i == rows.size() - 1)

{

rows.add(row);

break;

}

}

}

}

Map map = new HashMap();

for (Row row : this.getRows())

{

map.clear();

for (Event event : this.getTimeline())

{

if (event.getProcessName().equals(row.getProcessName()))

{

if (map.containsKey(event.getProcessName()))

{

int w = event.getStartTime() - (int) map.get(event.getProcessName());

row.setWaitingTime(row.getWaitingTime() + w);

}

else

{

row.setWaitingTime(event.getStartTime() - row.getArrivalTime());

}

map.put(event.getProcessName(), event.getFinishTime());

}

}

row.setTurnaroundTime(row.getWaitingTime() + row.getBurstTime());

}

}

}

1. **Shortest Job First:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class ShortestJobFirst extends CPUScheduler

{

@Override

public void process()

{

Collections.sort(this.getRows(), (Object o1, Object o2) -> {

if (((Row) o1).getArrivalTime() == ((Row) o2).getArrivalTime())

{

return 0;

}

else if (((Row) o1).getArrivalTime() < ((Row) o2).getArrivalTime())

{

return -1;

}

else

{

return 1;

}

});

List<Row> rows = Utility.deepCopy(this.getRows());

int time = rows.get(0).getArrivalTime();

while (!rows.isEmpty())

{

List<Row> availableRows = new ArrayList();

for (Row row : rows)

{

if (row.getArrivalTime() <= time)

{

availableRows.add(row);

}

}

Collections.sort(availableRows, (Object o1, Object o2) -> {

if (((Row) o1).getBurstTime() == ((Row) o2).getBurstTime())

{

return 0;

}

else if (((Row) o1).getBurstTime() < ((Row) o2).getBurstTime())

{

return -1;

}

else

{

return 1;

}

});

Row row = availableRows.get(0);

this.getTimeline().add(new Event(row.getProcessName(), time, time + row.getBurstTime()));

time += row.getBurstTime();

for (int i = 0; i < rows.size(); i++)

{

if (rows.get(i).getProcessName().equals(row.getProcessName()))

{

rows.remove(i);

break;

}

}

}

for (Row row : this.getRows())

{

row.setWaitingTime(this.getEvent(row).getStartTime() - row.getArrivalTime());

row.setTurnaroundTime(row.getWaitingTime() + row.getBurstTime());

}

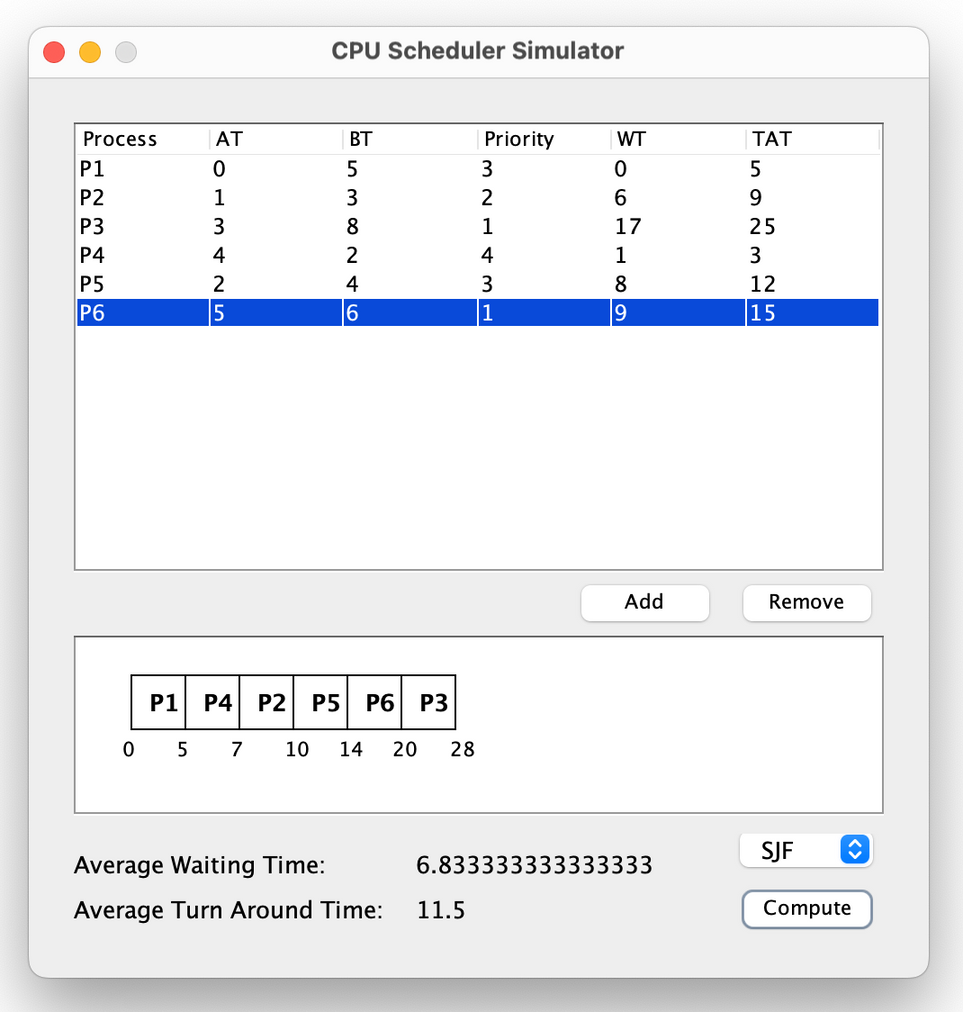
**OUTPUT**

1. **FCFS:**

A screenshot of a computer

Description automatically generated

1. **SJF:**

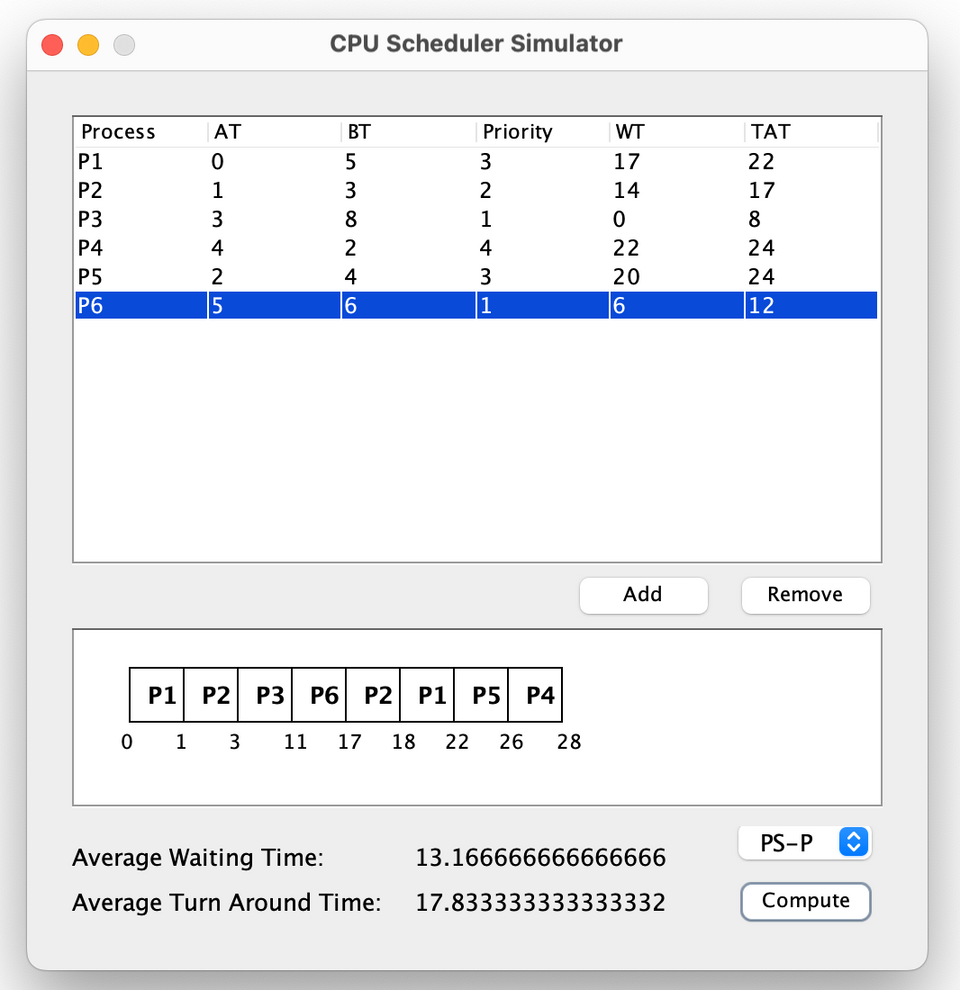
****

1. **Priority Scheduling - Non Preemptive:**

**A screenshot of a computer

Description automatically generated**

1. **Priority Scheduling – Preemptive**

****

1. **Round Robin:**

**A screenshot of a computer

Description automatically generated**

A screenshot of a computer

Description automatically generated

**CHAPTER 5**

**CONCLUSION**

In conclusion, the process scheduler project for operating systems represents a significant milestone in understanding and improving the management of system resources and the execution of processes. Through this project, we have delved into the intricate world of CPU scheduling algorithms, shedding light on their impact on system performance, efficiency, and responsiveness.

The project has equipped us with valuable insights into various scheduling algorithms, such as First-Come-First-Serve (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin. We've explored both non-preemptive and preemptive variants, each with its own unique characteristics and applications. These algorithms play a pivotal role in ensuring that the CPU is allocated efficiently to processes, ultimately influencing a system's ability to handle concurrent tasks and meet the demands of diverse workloads.

Our exploration of scheduling algorithms has highlighted the trade-offs that exist in the realm of process management. While some algorithms prioritize fairness and equitable access to system resources, others aim to optimize throughput and response times, making it crucial to choose the most suitable algorithm for a given context.

Furthermore, our project has emphasized the importance of accurate burst time estimation, particularly in preemptive scheduling, as well as the need for effective priority management in priority-based scheduling. We have also recognized the role of time quantum in the Round Robin algorithm, where the choice of this parameter can significantly impact the overall system performance.

As we conclude this project, we acknowledge that the world of process scheduling is not static. Advances in computing technology, evolving system requirements, and changing user expectations continue to shape the landscape of scheduling algorithms. Therefore, our project is not just a culmination but also a stepping stone to further research and innovation in the field.

In the ever-evolving domain of operating systems, the knowledge and experience gained from this project will be invaluable. It equips us to make informed decisions when selecting scheduling algorithms, optimizing their parameters, and addressing the dynamic needs of modern computing environments. The insights gathered during this project are essential not only for understanding the past and present but also for shaping the future of process scheduling in operating systems.

**CHAPTER 6**

**FUTURE SCOPE**

The future scope of process scheduler simulation holds immense potential as technological advancements continue to shape the landscape of computing systems. One prominent avenue of exploration is the integration of machine learning and artificial intelligence into scheduling algorithms, enabling adaptive and intelligent decision-making processes that can enhance efficiency in dynamic computing environments. The advent of quantum computing introduces a novel dimension to scheduling challenges, necessitating the development of simulation models tailored to the distinctive requirements of quantum processes. As edge computing gains prominence, future simulations are likely to address the intricacies of scheduling in decentralized environments, accommodating resource constraints and optimizing for low-latency distributed processing. The convergence of traditional algorithms with machine learning components in hybrid strategies presents another area of interest, fostering more responsive and adaptive scheduling mechanisms. With a growing emphasis on sustainability, energy-aware scheduling simulations are anticipated to prioritize algorithms that minimize environmental impact. Security-aware scheduling mechanisms may also emerge as a focus, aligning with evolving cybersecurity concerns. Furthermore, future simulations might explore predictive models for dynamic workload and resource variations, offering insights into anticipatory scheduling strategies. The integration of human-in-the-loop elements and the consideration of user-centric criteria, as well as cross-layer optimization approaches, are poised to shape the next frontier of process scheduler simulation, contributing to the evolution of efficient and intelligent scheduling in the dynamic landscape of computing systems.

In addition to the aforementioned areas, the future of process scheduler simulation is likely to witness increased attention on optimizing scheduling for emerging technologies such as blockchain and distributed ledger systems. These decentralized computing environments present unique challenges related to consensus algorithms and distributed processing, prompting researchers to explore how scheduling mechanisms can be refined to accommodate the intricacies of these evolving technologies. Furthermore, a growing emphasis on cross-layer optimization is expected to influence the development of scheduling strategies that holistically consider interactions with various system layers, including memory management, storage, and network communication. This integrated approach seeks to enhance overall system performance by addressing interdependencies between different components.

**CHAPTER 7**

**REFERENCES**

**"Modern Operating Systems" by Andrew S. Tanenbaum**

* This classic book covers a wide array of process scheduling algorithms, offering in-depth explanations of Round Robin, First Come First Serve, and Priority Scheduling.

**"Operating System Concepts" by Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne**

* Provides historical context and evolution of process scheduling, offering foundational knowledge for understanding the subject.

**"Comparison of Different Scheduling Algorithms in the Linux Kernel" by Wencheng Lu and Xiaohui Kang**

* A research paper offering insights into practical implementations of scheduling policies, useful for real-world applications.

**IEEE Transactions on Computers and ACM Transactions on Computer Systems**

* These reputable journals feature cutting-edge research articles on novel process scheduling techniques and advancements in the field.

**USENIX Annual Technical Conference**

* Explore this conference's papers for practical applications and challenges related to process scheduling.

**Linux Kernel Documentation**

* Valuable for understanding the implementation details of scheduling algorithms in the Linux operating system.

**"Operating Systems: Internals and Design Principles" by William Stallings**

* Stallings' book covers various aspects of operating systems, including process scheduling, and serves as a comprehensive reference for understanding the internal mechanisms.

**Official Documentation of Process Scheduling in UNIX/Linux**

* Refer to official documentation for UNIX/Linux operating systems to gain precise insights into how process scheduling is implemented in these widely used environments.

**"Multilevel Feedback Queue Scheduling in Real-Time Systems" by Mark Stanovich**

* This research paper delves into the intricacies of multilevel feedback queue scheduling, offering specialized knowledge for those interested in real-time systems and advanced scheduling techniques.

**APPENDIX**

An appendix for a process scheduler simulation document typically includes supplementary information, details, or artifacts that support and enhance the main content of the document. Here's a suggested structure for an appendix on process scheduler simulation:

Appendix A: Simulation Parameters and Settings

This section provides a detailed list of the parameters and settings used in the process scheduler simulation. It includes information on time quantum, priority levels, scheduling algorithms employed, and any other configuration specifics. Researchers and practitioners can refer to this appendix to understand the simulation environment and reproduce the results.

Appendix B: Simulation Code Snippets

For transparency and reproducibility, consider including snippets of the simulation code used in the research. This can be particularly helpful for readers who may want to implement or adapt the simulation for their own studies. Ensure that the code snippets are well-commented and accompanied by explanatory notes.

Appendix C: Performance Metrics Definitions

Outline the definitions and calculations of the performance metrics used to evaluate the effectiveness of the process scheduler simulation. This section helps readers understand how metrics like CPU utilization, throughput, response time, and others were measured and interpreted in the study.

Appendix D: Raw Simulation Data

Include raw data sets or logs generated during the simulation runs. This allows readers to conduct additional analyses, verify the presented results, or explore alternative performance metrics. Ensure that the data is well-organized, with clear labels and annotations.

Appendix E: Visualization of Results

Present visual representations of the simulation results, such as graphs, charts, or tables. These visualizations can offer a concise and accessible overview of the key findings. Label the figures appropriately and refer to them in the main text when necessary.

Appendix F: Sensitivity Analysis

If applicable, include a sensitivity analysis that explores how changes in simulation parameters or initial conditions impact the results. This helps to assess the robustness of the findings and provides insights into the system's behavior under different scenarios.

Appendix G: Survey Questionnaire (if applicable)

In case the study involves user feedback or a survey component related to the simulation, include the survey questionnaire used. This helps readers understand the user perspective and any qualitative data collected during the research.

Appendix H: Ethical Considerations

Provide information on the ethical considerations, approvals, and guidelines followed during the simulation study. If the study involved human subjects or sensitive data, outline the steps taken to ensure ethical research practices were upheld.

Appendix I: Glossary of Terms

Include a glossary that defines and explains technical terms, acronyms, or specialized terminology used throughout the document. This section aids readers in better understanding the content without having to refer to external sources.

Remember to refer to these appendices appropriately in the main text, and ensure that each appendix is clearly titled and paginated for ease of navigation.

Appendix J: Simulation Tool Information

Provide details about the simulation tool or framework used in the study. Include information on the version, features, and any customizations made for the process scheduler simulation. This appendix helps readers comprehend the simulation environment and promotes transparency in the research methodology.

Appendix K: Comparative Analysis with Real-World Data

If applicable, include a section that compares the simulation results with real-world data, showcasing the model's accuracy and relevance to practical scenarios. This can involve datasets from actual systems or industry benchmarks, providing additional validation to the simulation outcomes.

Appendix L: User Manual for Simulation Replication

Offer a comprehensive user manual or guide that outlines step-by-step instructions for replicating the process scheduler simulation. This includes details on software dependencies, installation procedures, and any prerequisites needed to recreate the study. Enhancing reproducibility ensures the credibility of the research.

Appendix M: Challenges and Limitations

Discuss challenges faced during the simulation process and any inherent limitations in the study. This section provides a transparent account of the research constraints and offers insights for future improvements or adaptations of the simulation model.