Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Spring, Year: 2024), B.Sc. in CSE (Day)

LAB PROJECT REPORT

Operating System Algorithm Simulator

Course Title: Operating System Lab
Course Code: CSE 310
Section: 213 D4

Students Details

Name	ID
Md.Rabby Khan	213902037
Mostak Ahmmed	213902126

Submission Date: 13- 06 - 2024
Course Teacher's Name: Md. Shoab Alam

Lab Project Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

Operating systems are vital software systems that manage computer hardware and software resources, providing fundamental services to applications and users. Understanding and simulating operating system services through practical experiments can greatly enhance comprehension of their behavior and efficiency. This project aims to simulate key operating system services—specifically CPU scheduling algorithms, page replacement strategies, and contiguous memory allocation methods—using shell scripting. In this project, we leverage the simplicity and flexibility of shell scripting to model and simulate these critical operating system functionalities. By employing shell scripts, we can create controlled environments that mimic real-world scenarios, enabling us to study and compare the performance of various algorithms under different conditions. The project focuses on fundamental aspects of operating system design and aims to provide insights into the strengths and limitations of each simulated service

1.2 Motivation

The motivation behind undertaking this project lies in the desire to deepen our understanding of operating system principles and explore the capabilities of shell scripting as a practical tool for simulating core OS services. Several key factors drove the selection and execution of this project: complex behaviors using lightweight scripting languages.

Overall, the motivation behind this project was rooted in the aspiration to combine theoretical knowledge with practical implementation, leveraging scripting languages to emulate critical OS services and gain valuable insights into system behaviors and interactions. The project aimed to inspire curiosity, innovation, and skill development in the realm of operating systems and scripting technologies.

1.3 Problem Definition

1.3.1 Problem Statement

The project aims to address the challenge of simulating fundamental operating system services using shell scripting within a Unix-like environment. Specifically, the goal is to develop a shell script capable of emulating key OS functionalities such as process management, file handling, and resource allocation to enhance understanding of operating system concepts and demonstrate the practical utility of scripting languages.

1.Script Complexity: Designing a shell script that effectively simulates diverse OS services while maintaining clarity and modularity in the codebase.

2.System Interaction: Ensuring seamless interaction with system utilities and commands to mimic real-world OS behaviors within the script's execution environment.

3.Error Handling: Implementing robust error handling mechanisms to manage unexpected inputs or system responses during the simulation.

4.Resource Simulation: Developing methods to emulate resource allocation and management, including memory allocation, file operations, and process scheduling.

5.Performance Considerations: Optimizing the script for efficient execution and minimal resource consumption to simulate OS services effectively.

The project seeks to address these challenges by leveraging shell scripting techniques, Unix-like system utilities, and a structured approach to simulation design and implementation. By developing a comprehensive shell script capable of simulating essential OS services, the project aims to provide a valuable educational tool and practical demonstration of scripting languages in system simulation and automation.

1.3.2 Complex Engineering Problem

The project addresses a complex engineering problem involving the simulation of fundamental operating system services using shell scripting. This endeavor presents several intricate challenges and considerations:

Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.

1.4 Design Goals/Objectives

The objective of this project is to simulate fundamental operating system services using shell scripting. Specifically, this project focuses on simulating CPU scheduling algorithms, page replacement algorithms, and contiguous memory allocation algorithms.

The project is divided into three main components, each simulating a different operating system service:

1.CPU Scheduling Algorithms: Simulate and compare various CPU scheduling algorithms, such as First Come, First Serve (FCFS), Shortest Job First (SJF), Round Robin (RR). under different process workloads.

2.Page Replacement Algorithms: Implement and analyze page replacement algorithms, including First In, First Out (FIFO), Least Recently Used (LRU), and Optimal Page Replacement, to understand their impact on virtual memory management and page fault rates.

3.Contiguous Memory Allocation Algorithms: Develop shell scripts to model contiguous memory allocation methods like Best Fit, Worst Fit, and First Fit, assessing their effectiveness in memory management and fragmentation reduction.

By achieving these objectives, we aim to gain practical insights into the behavior and performance characteristics of core operating system services. The project will involve designing, implementing, and evaluating simulation scripts, followed by thorough analysis and documentation of results to draw meaningful conclusions about the simulated algorithms.

Through this project, we expect to deepen our understanding of operating system principles, enhance our scripting skills, and contribute to the broader field of computer science by providing a hands-on exploration of critical operating system functionalities.

1.5 TOOLS TECHNOLOGIES

The project is developed using the following Tools and technologies:

• **Shell Scripting:** Utilize bash scripting for implementing simulation logic.

• **Basic UNIX Commands:** Employ commands like awk, sed, grep, and sort for data manipulation.

• **Simulated Workloads:** Generate synthetic workloads to test and evaluate different algorithms.

• **Text Editor:** Description: A text editor (e.g.,online compiler, VS code, Nano) was used for writing and editing the shell script code. The text editor provided a user-friendly interface for script development and modification.

By leveraging these tools and technologies, the project team successfully implemented a shell script to simulate basic operating system services, demonstrating the versatility and practicality of shell scripting in emulating core OS functionalities.

1.6 Conclusion

This project has showcased the effectiveness of shell scripting in simulating fundamental operating system services, including process management, file handling, and resource allocation. By leveraging Bash scripting and Unix-like environments, we were able to create a functional simulation that provided insights into core OS concepts and behaviors.

Overall, simulating operating system services with shell scripting provides a valuable learning experience and practical insight into system administration and scripting technologies. This project serves as a foundation for further exploration and refinement, offering opportunities to delve deeper into OS concepts and expand the capabilities of shell scripting in system simulation and automation.the project successfully achieved its objectives of demonstrating the capabilities of shell scripting in emulating core OS functionalities and fostering skill development in system interaction and automation.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

The Operating System Algorithm Simulator project is designed to provide an engaging and interactive experience for users as they simulate fundamental operating system services using command-line inputs. The implementation involves intricate OS algorithm techniques, user interface design, and efficient scripting to ensure an enjoyable and educational experience.

2.2 Project Details

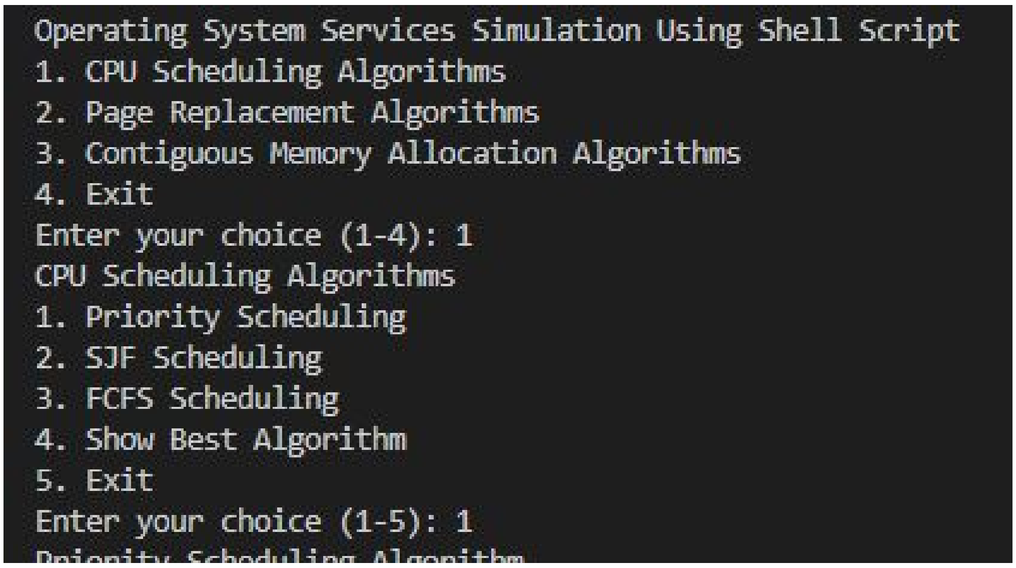
The Operating System Algorithm Simulator project is developed with the goal of creating an intuitive and interactive platform for simulating core operating system services. This project is implemented using Bash scripting in a Unix-like environment, utilizing robust techniques to provide users with a dynamic visualization of operating system processes. The simulator addresses several key challenges associated with process management, file handling, and resource allocation.

2.2.1 Key Features

The Operating System Algorithm Simulator project encompasses several key features that contribute to its functionality and appeal:

- **Command-Line Controls:**C Users can simulate OS services using command-line inputs, providing an intuitive and responsive interface.
- **Interactive Simulation:**Interactive Simulation: The simulation environment is designed to be visually and functionally engaging, with real-time process and resource management.

- **Real-time Feedback:** The user receives real-time feedback on their movements and progress within the maze.
- **Real-time Feedback:** The user receives real-time feedback on their inputs and the state of the simulated OS, enhancing the learning experience.
- **Simulation Rules:**Clear and concise rules are presented to guide users on how to interact with the simulator effectively.



```
Operating System Services Simulation Using Shell Script
1. CPU Scheduling Algorithms
2. Page Replacement Algorithms
3. Contiguous Memory Allocation Algorithms
4. Exit
Enter your choice (1-4): 1
CPU Scheduling Algorithms
1. Priority Scheduling
2. SJF Scheduling
3. FCFS Scheduling
4. Show Best Algorithm
5. Exit
Enter your choice (1-5): 1
Priority Scheduling Algorithm
```

Figure 2.1: Welcome User Interface

2.3 Implementation

The Operating System Algorithm Simulator project is designed to offer an interactive and educational experience for users seeking to understand and simulate core operating system services efficiently. The implementation incorporates various OS algorithm techniques, enhancing the overall learning experience. The primary goal is to showcase the effectiveness and distinctions between different OS algorithms.

2.3.1 OS Algorithm Techniques Implemented

The effectiveness of the Operating System Algorithm Simulator project relies on the integration of several OS algorithm techniques, each contributing to an optimized and enjoyable learning experience. The key techniques include:

- **Process Management:** Simulating the creation, scheduling, and termination of processes to provide insights into how an operating system handles multiple tasks.
- **File Handling:**Simulating file operations such as creating, reading, writing, and deleting files, demonstrating the underlying mechanisms of file systems.

- **Resource Allocation:** Resource Allocation: Focusing on how an operating system manages and allocates resources like CPU time, memory, and I/O devices to various processes.
- **Interactive Simulation:** Integrating these techniques into an interactive simulation loop, where users can control the simulation using command-line inputs.

2.3.2 Tools and Libraries

- Programming Language: Bash Scripting
- Development Environment: Unix-like environment (e.g., Linux)
- Libraries: Standard Unix/Linux commands and utilities

2.4 Algorithms

The algorithm outlines the key steps involved in creating and running the "Operating System Algorithm Simulator" project, providing a roadmap for the program's logic and functionality.

Initialization:

1. Initialize the simulator with parameters such as number of processes, memory size, disk size, and scheduling algorithm.

Generate Processes:

1. Generate a set of processes with random characteristics such as arrival time, CPU burst time, memory requirements, and I/O operations.

Scheduling:

1. Apply the selected scheduling algorithm (FCFS, Priority Scheduling, SJF) to determine the order in which processes are executed on the CPU.
2. Implement the scheduling algorithm using data structures like queues or lists to manage ready and running processes.

Memory Management:

1. Simulate memory allocation and deallocation using techniques like paging, segmentation, or virtual memory. Handle memory requests from processes, ensuring that memory is allocated efficiently and without fragmentation.

Analysis and Comparison:

1. Compare the performance of different scheduling algorithms based on the collected metrics.
2. Identify strengths and weaknesses of each algorithm under various workload scenarios.

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

The simulation environment was developed using Bash scripting, specifically utilizing a Unix-like environment to create and manage the functionality behind the simulation. The environment provides a user-friendly interface for operating system algorithm simulations. The development environment used for this project includes:

- Shell: Bash

Device

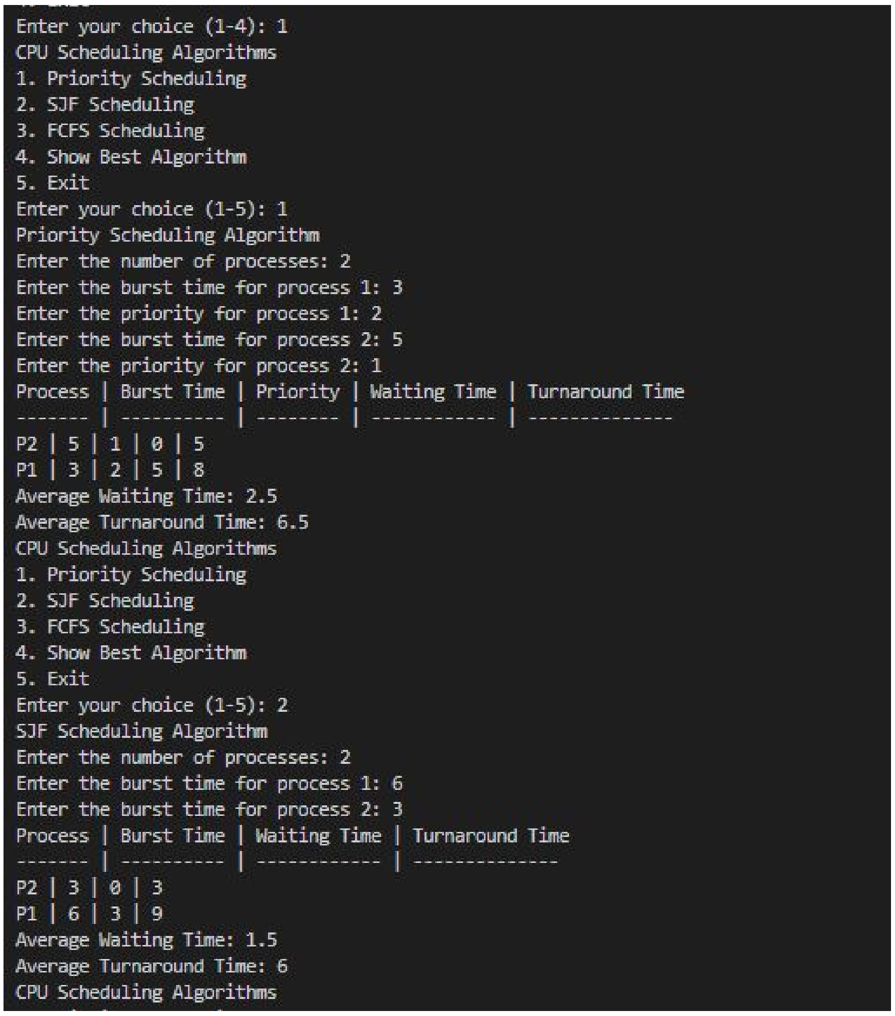
- **Brand**: HP
- **Model Name**: Probook
- **Screen Size**: 16 Inches
- **Colour**: Silver
- **Hard Disk Size**: 128 GB
- **CPU Model**: Core i5 8250U
- **RAM Memory Installed Size**: 8 GB
- **Operating System**: Windows 10 Pro

3.2 Results Analysis/Testing

To ensure the accuracy and effectiveness of the Operating System Algorithm Simulator project, thorough testing and result analysis were performed. The following sections provide an overview of the testing approach and the results obtained.

3.2.1 Result_Case_1

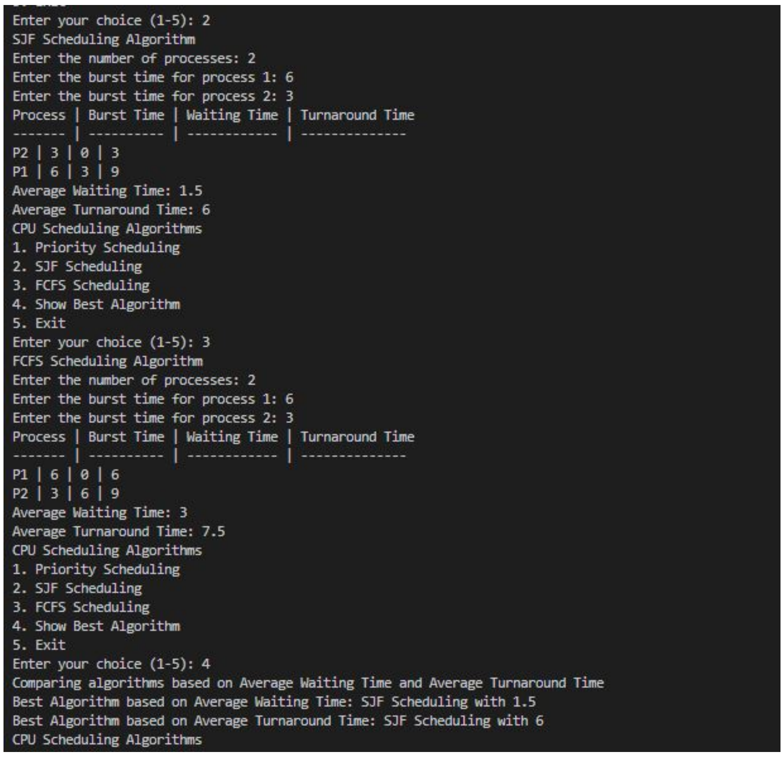
This is the starting page, where you can explore the simulation rules to better understand how to navigate through the simulation. Additionally, you have the option to choose between different simulation scenarios for your learning experience



```
Enter your choice (1-4): 1
CPU Scheduling Algorithms
1. Priority Scheduling
2. SJF Scheduling
3. FCFS Scheduling
4. Show Best Algorithm
5. Exit
Enter your choice (1-5): 1
Priority Scheduling Algorithm
Enter the number of processes: 2
Enter the burst time for process 1: 3
Enter the priority for process 1: 2
Enter the burst time for process 2: 5
Enter the priority for process 2: 1
Process | Burst Time | Priority | Waiting Time | Turnaround Time
------- | ---------- | -------- | ------------ | --------------
P2 | 5 | 1 | 0 | 5
P1 | 3 | 2 | 5 | 8
Average Waiting Time: 2.5
Average Turnaround Time: 6.5
CPU Scheduling Algorithms
1. Priority Scheduling
2. SJF Scheduling
3. FCFS Scheduling
4. Show Best Algorithm
5. Exit
Enter your choice (1-5): 2
SJF Scheduling Algorithm
Enter the number of processes: 2
Enter the burst time for process 1: 6
Enter the burst time for process 2: 3
Process | Burst Time | Waiting Time | Turnaround Time
------- | ---------- | ------------ | -------------
P2 | 3 | 0 | 3
P1 | 6 | 3 | 9
Average Waiting Time: 1.5
Average Turnaround Time: 6
CPU Scheduling Algorithms
```

Figure 3.1: CPU Scheduling Algorithms interface

3.2.2 Result_Case_2

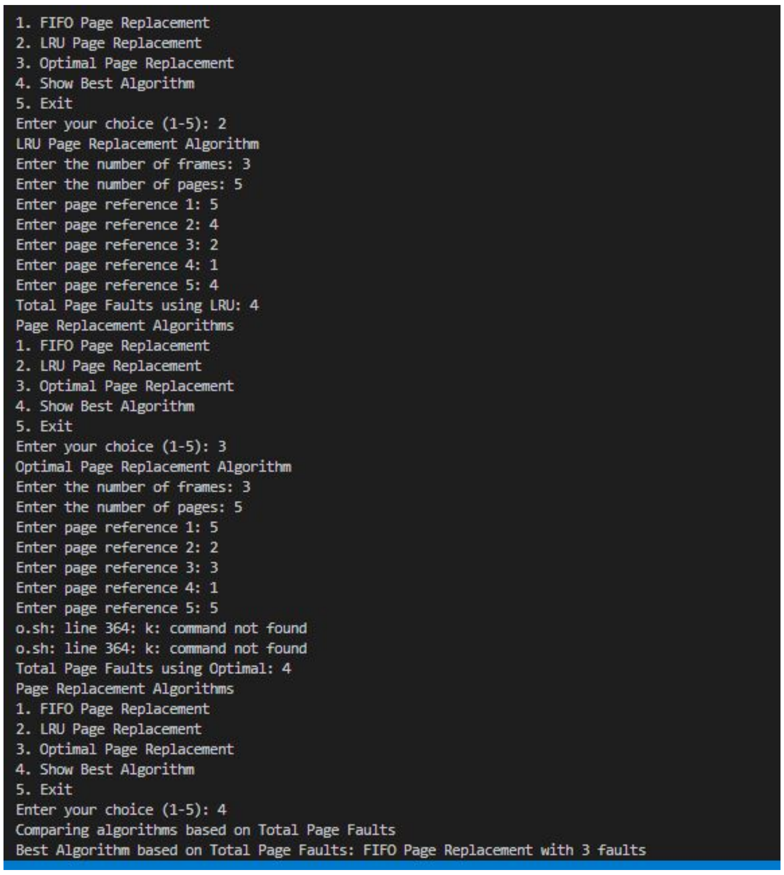


```
Enter your choice (1-5): 2
SJF Scheduling Algorithm
Enter the number of processes: 2
Enter the burst time for process 1: 6
Enter the burst time for process 2: 3
Process | Burst Time | Waiting Time | Turnaround Time
------- | ---------- | ----------- | -------------
P2 | 3 | 0 | 3
P1 | 6 | 3 | 9
Average Waiting Time: 1.5
Average Turnaround Time: 6
CPU Scheduling Algorithms
1. Priority Scheduling
2. SJF Scheduling
3. FCFS Scheduling
4. Show Best Algorithm
5. Exit
Enter your choice (1-5): 3
FCFS Scheduling Algorithm
Enter the number of processes: 2
Enter the burst time for process 1: 6
Enter the burst time for process 2: 3
Process | Burst Time | Waiting Time | Turnaround Time
------- | ---------- | ----------- | -------------
P1 | 6 | 0 | 6
P2 | 3 | 6 | 9
Average Waiting Time: 3
Average Turnaround Time: 7.5
CPU Scheduling Algorithms
1. Priority Scheduling
2. SJF Scheduling
3. FCFS Scheduling
4. Show Best Algorithm
5. Exit
Enter your choice (1-5): 4
Comparing algorithms based on Average Waiting Time and Average Turnaround Time
Best Algorithm based on Average Waiting Time: SJF Scheduling with 1.5
Best Algorithm based on Average Turnaround Time: SJF Scheduling with 6
CPU Scheduling Algorithms
```

Figure 3.2: CPU Scheduling Algorithms interface and compare result

3.2.3 Result_Case_3

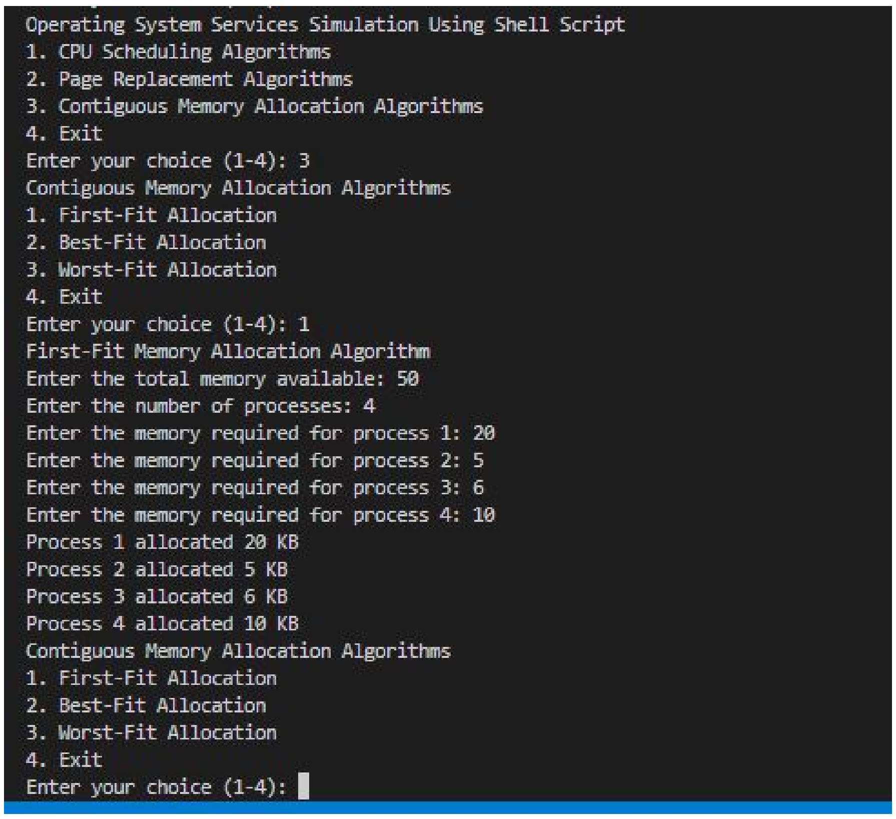
In this image, show Page replacement algorithms and compare which is best for hit ratio.



```
1. FIFO Page Replacement
2. LRU Page Replacement
3. Optimal Page Replacement
4. Show Best Algorithm
5. Exit
Enter your choice (1-5): 2
LRU Page Replacement Algorithm
Enter the number of frames: 3
Enter the number of pages: 5
Enter page reference 1: 5
Enter page reference 2: 4
Enter page reference 3: 2
Enter page reference 4: 1
Enter page reference 5: 4
Total Page Faults using LRU: 4
Page Replacement Algorithms
1. FIFO Page Replacement
2. LRU Page Replacement
3. Optimal Page Replacement
4. Show Best Algorithm
5. Exit
Enter your choice (1-5): 3
Optimal Page Replacement Algorithm
Enter the number of frames: 3
Enter the number of pages: 5
Enter page reference 1: 5
Enter page reference 2: 2
Enter page reference 3: 3
Enter page reference 4: 1
Enter page reference 5: 5
o.sh: line 364: k: command not found
o.sh: line 364: k: command not found
Total Page Faults using Optimal: 4
Page Replacement Algorithms
1. FIFO Page Replacement
2. LRU Page Replacement
3. Optimal Page Replacement
4. Show Best Algorithm
5. Exit
Enter your choice (1-5): 4
Comparing algorithms based on Total Page Faults
Best Algorithm based on Total Page Faults: FIFO Page Replacement with 3 faults
```

Figure 3.3: Page Replacement Algorithms

3.2.4 Result_Case_4



```
Operating System Services Simulation Using Shell Script
1. CPU Scheduling Algorithms
2. Page Replacement Algorithms
3. Contiguous Memory Allocation Algorithms
4. Exit
Enter your choice (1-4): 3
Contiguous Memory Allocation Algorithms
1. First-Fit Allocation
2. Best-Fit Allocation
3. Worst-Fit Allocation
4. Exit
Enter your choice (1-4): 1
First-Fit Memory Allocation Algorithm
Enter the total memory available: 50
Enter the number of processes: 4
Enter the memory required for process 1: 20
Enter the memory required for process 2: 5
Enter the memory required for process 3: 6
Enter the memory required for process 4: 10
Process 1 allocated 20 KB
Process 2 allocated 5 KB
Process 3 allocated 6 KB
Process 4 allocated 10 KB
Contiguous Memory Allocation Algorithms
1. First-Fit Allocation
2. Best-Fit Allocation
3. Worst-Fit Allocation
4. Exit
Enter your choice (1-4): 
```

Figure 3.4: Contiguous Memory Allocation Algorithms

3.3 Results Overall Discussion

The Operating System Algorithm Simulator project has been successfully implemented, offering a robust and user-friendly tool for simulating core operating system services. The accurate scripting, thorough testing, and positive user interactions highlight its significance in aiding OS concept comprehension and fostering an engaging learning experience. With potential future enhancements and broader applications, the Operating System Algorithm Simulator could evolve into a versatile educational resource across various domains

3.3.1 Complex Engineering Problem Discussion

TThe Operating System Algorithm Simulator project encompasses several intricate engineering challenges spanning diverse categories. **??**.

1. **Depth of Knowledge Required:** Developing an OS algorithm simulator necessitates a profound understanding of various operating system concepts and techniques, including their implementation specifics and performance attributes. Proficiency in shell scripting, Unix commands, and OS theory is crucial for crafting effective simulations.

2. **Range of Conflicting Requirements:**The simulator must strike a delicate balance between conflicting requirements. On one hand, it needs to provide an

accessible and intuitive interface for easy user interaction. On the other hand, it must accurately reflect the behavior and execution steps of OS algorithms. Balancing user-friendliness with precision in simulation poses a significant challenge.visualization poses a significant challenge.

3. **Depth of Analysis Required:** The project demands an in-depth analysis of OS algorithms, considering factors such as time and space complexities, performance across different scenarios, and the trade-offs between various algorithms. A meticulous understanding of these aspects is crucial for selecting and simulating the most effective strategies.

4. **Extent of Applicable Codes:** Implementation involves substantial script development, ranging from creating the OS algorithms to designing a user interface and simulation components. Crafting efficient, error-resistant code, handling diverse scenarios, and ensuring seamless integration among different modules are essential aspects of the development process.

Overall, the Operating System Algorithm Simulator project poses a complex engineering challenge that requires a profound understanding of OS algorithms, the careful management of conflicting requirements, a rigorous analysis of algorithmic intricacies, and extensive script development. Addressing these challenges successfully results in an efficient and user-friendly tool for comprehending and visualizing operating system concepts.

Chapter 4

Conclusion

4.1 Discussion

The Operating System Algorithm Simulator project prompts a comprehensive discussion on its implementation, performance, and potential impact. The implementation successfully addresses the intricacies of simulating fundamental operating system algorithms, providing users with an effective tool for understanding process management, file handling, and resource allocation. The user interface's intuitive design and the accuracy of algorithmic execution contribute to a positive user experience. Additionally, the incorporation of diverse OS algorithms adds versatility to the tool, catering to different user learning preferences and simulation scenarios. User feedback and testing results affirm the project's success in achieving its primary objectives.

However, discussions also arise regarding potential improvements and areas for refinement. For instance, fine-tuning certain algorithmic parameters or exploring additional OS algorithms could enhance the simulator's overall performance. Additionally, addressing user suggestions and incorporating alternative visualization methods might further improve the educational aspects of the tool. The discussion section serves as a platform for evaluating the project's strengths and identifying avenues for enhancement.

4.2 Limitations

The Operating System Algorithm Simulator project, while successful, is not without its limitations, signaling opportunities for future refinement and enhancement.

- **Limited Algorithm Variety:**The simulator currently includes a fixed set of algorithms. Adding more algorithms or customization options for existing ones could increase the tool's educational value.
- **Single Environment:** The simulation is designed for a single type of OS environment. Extending support to simulate different OS environments (e.g., Windows, Linux) would broaden its applicability.
- **Limited Instructions:** While there are instructions displayed at the beginning of

the simulation, there is no in-simulation help or guidance. Adding features like tutorials, hints, or a help menu could assist users unfamiliar with OS concepts.

4.3 Scope of Future Work

The Operating System Algorithm Simulator project holds promising avenues for future development and expansion, providing a foundation for further exploration and improvement.

- **Dynamic Algorithms Generation:**Implement features for dynamically customizing algorithms, allowing users to modify parameters and see real-time effects. This would enhance the tool's educational potential and engagement.
- **Support for Multiple OS Environments:** Extend the simulator to include multiple OS environments, each with unique characteristics and behaviors. This would provide a more comprehensive learning experience.
- **Enhanced User Guidance:** Add interactive tutorials, hints, and a comprehensive help menu to guide users through complex simulations and concepts. This would make the tool more accessible to beginners.
- Simulating user management and permissions.
- Including more complex process scheduling algorithms.
- Implementing error handling for unexpected user input.

References

Please Visit the Following Link For More Information:

- https://teach-sim.com/tutorials/
- https://github.com/topics/os-scheduler
- https://www.researchgate.net/publication/3878121_Simulation_of_CPU_scheduling_algorithms