CPU Scheduler Simulation Report

Basma ARNAOUI - Omar Alfarouq BOUHADI

April 30, 2024

Abstract

This report details the development and implementation of a CPU Scheduler Simulation tool designed to analyze the performance of various CPU scheduling algorithms. The tool aims to simulate First-Come, First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin (RR), providing insights into their operational efficiencies through a user-friendly interface.

Contents

1 Introduction

CPU scheduling is a crucial aspect of operating systems, responsible for managing the allocation of CPU resources among processes. In this project, we develop a CPU Scheduler Simulation tool to analyze the performance of different scheduling algorithms. The tool enables users to simulate and compare the effectiveness of algorithms such as First-Come, First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin (RR).

2 Objectives

The primary objectives of this project include:

- Simulating multiple CPU scheduling algorithms.
- Comparing the effectiveness of these algorithms through performance metrics such as turnaround time, waiting time, and CPU utilization.
- Providing a user-friendly interface that allows users to customize inputs and visualize the scheduling process effectively.

3 Methodology

3.1 Tech stack choice: Python + Flask

The selection of Flask, a lightweight web framework for Python, as the cornerstone of our CPU Scheduler Simulation project was guided by a strategic blend of factors aligning with our project's objectives and requirements. Python's versatility and simplicity rendered it an optimal choice for swiftly prototyping and implementing the intricate logic necessitated by CPU scheduling algorithm simulations. The extensive library support within Python facilitated seamless integration of various components, empowering us to efficiently manipulate data, visualize results, and develop a user-friendly web interface. Meanwhile, Flask's minimalist design and scalability enabled us to maintain a lean codebase, ensuring swift development and enhancing the project's performance. Leveraging Flask's modular architecture and well-documented features, we could rapidly iterate on functionalities, address issues, and adapt to evolving project needs. Moreover, the widespread adoption of Python and Flask within the developer community provided invaluable resources, tutorials, and community support, enabling us to overcome challenges and adhere to best practices throughout the project lifecycle. By harnessing the combined strengths of Flask and Python, we delivered a robust CPU Scheduler Simulation tool that not only analyzes scheduling algorithms' performance effectively but also lays a solid groundwork for future enhancements and expansions.

3.2 System Architecture

The system architecture consists of three main components: input handling, algorithm implementation, and visualization. The input handling component parses user inputs, including process details and selected algorithms. The algorithm implementation component contains the logic for each scheduling algorithm, while the visualization component generates visualizations and performance metrics.

3.3 Algorithm Implementation

The implementation of each scheduling algorithm follows standard algorithms. For example, FCFS schedules processes based on their arrival times, SJF schedules the shortest job first, Priority Scheduling assigns priorities to processes, and RR allocates CPU time in time slices.

3.4 Code Explanation

We used Python programming language to implement the CPU scheduling algorithms due to its simplicity and availability of libraries for data visualization. The code snippets provided in the document illustrate the implementation of each algorithm. For instance, the FCFS algorithm code snippet demonstrates how processes are scheduled based on their arrival times.

4 Conclusion

In conclusion, the CPU Scheduler Simulation tool provides valuable insights into the performance of various scheduling algorithms. By simulating different scenarios and analyzing performance metrics, we gain a better understanding of the strengths and weaknesses of each algorithm. We chose Flask to build the project due to its lightweight nature and simplicity in creating web applications. Future work may involve expanding the tool to include more scheduling algorithms and incorporating real-time scheduling scenarios.