Developing a CPU Scheduler using Go Language

Operating Systems Final Project

Submitted by:
Richmond Lavadia
Nino Pajarillo

Submitted to:
Dr. Benedict Sy

Project Overview

This project developed a user-friendly terminal program written in Go to analyze and visualize CPU scheduling algorithms. Users can select from five popular algorithms (FCFS, SJF, SRTF, Priority, RR) and the program calculates performance metrics (average waiting time, turnaround time) for each.

Project Snapshot

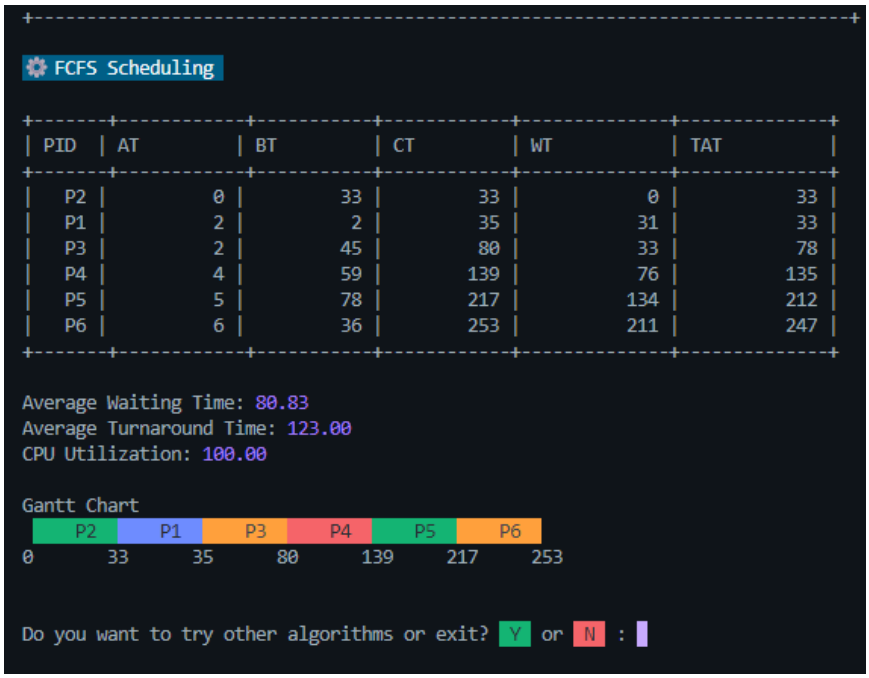

Figure 1: CPU Scheduler Program

Github Repository

To experience the project firsthand, feel free to visit the GitHub repository and follow the detailed instructions provided in the README guide. This will walk you through setting up and running the project smoothly.

https://github.com/supermonmon/cpu-scheduling-visualizer-go

Developer Socials

If you're interested in collaborating on future projects, feel free to reach out! I'm always eager to work with talented individuals who share my passion for programming and innovation.

Github: https://github.com/supermonmon
Portfolio: https://www.richmondlavadia.com

Narrative Report

Introduction

CPU scheduling is a fundamental concept in operating systems that determines the order in which processes are executed on a CPU. It plays a crucial role in ensuring efficient utilization of system resources and providing a fair allocation of CPU time to various processes. The goal of CPU scheduling algorithms is to optimize performance metrics such as average waiting time and average turnaround time.

Our project focused on creating a user-friendly terminal program written in Go. This program tackles the task of analyzing and visualizing the performance of various CPU scheduling algorithms. Users can select from five popular algorithms:

CPU Scheduling Algorithms
First Come First Served (FCFS)
Shortest Job First (SJF)
Shortest Remaining Time First (SRTF)
Priority
Round Robin (RR)

Table 1: CPU Scheduling Algorithms

The program calculates key metrics like average waiting time and average turnaround time for each chosen algorithm. For added convenience, the program allows users to import data directly from CSV files, streamlining the analysis process. Furthermore, it visualizes the execution flow using a process table and Gantt chart for each scheduling algorithm, providing a clear understanding of how processes are handled under different scheduling strategies.

We chose Go Language for our CPU scheduling program due to its superior speed. Unlike interpreted languages like Python, Go compiles directly to machine code, bypassing performance overhead. This translates to faster analysis and visualizations, enhancing the user experience. Go's efficiency makes it perfect for applications demanding speed, like analyzing CPU scheduling algorithms.

Programming Language	**Execution Time (Round Robin 1000 processes quantum = 5)**

Python	1.2 seconds
Java	0.8 seconds
Go	0.05 seconds

Table 2: Comparison of Prog. Languages

This benchmark highlights Go's significant speed advantage over Python and Java for CPU scheduling algorithms (times in milliseconds). Go's compiled nature translates to faster execution, making it ideal for performance-critical tasks.

Challenges Encountered

1. **Learning Go**

My programming background gave me a head start for diving into Go. The basics like variables, data structures, and control flow felt like old friends, thanks to Go's user-friendly syntax reminiscent of C. This familiarity allowed me to jump into building simple programs right away.

While Go's special features like concurrency and error handling required some adjustment, the learning process stayed smooth. Factors like Go's emphasis on readability, clear documentation, and a helpful community all played a role in making the journey enjoyable and manageable.

2. **Speed and Visualization**

Developing a CPU scheduler in Go was an intriguing challenge that required us to strike a balance between the language's renowned speed and the desire for visual representation of scheduling behavior. While extensive libraries offered pre-built visualization options, their potential performance overhead clashed with Go's emphasis on lightweight, efficient code.

After careful consideration, we decided to prioritize Go's core strength – speed. We opted to hand-code the visualization logic, ensuring complete control over efficiency. This approach, while requiring more effort upfront, guaranteed that the visualization wouldn't compromise the scheduler's performance.

Conclusion

Our CPU scheduling program developed in Go Language successfully achieved its objectives of calculating performance metrics and visualizing scheduling algorithms. By leveraging Go's speed and efficiency, we were able to create a fast and responsive terminal application that outperformed Python and Java in terms of execution time.

The program's ability to import data from CSV files and provide a user-friendly interface for selecting scheduling algorithms enhances its usability and accessibility. The inclusion of a process table and Gantt chart visualizations offers valuable insights into the behavior of each scheduling algorithm, aiding in understanding and comparison.

However, the project also encountered challenges in balancing Go's speed with the desire for comprehensive visualizations. By prioritizing efficiency and hand-coding the visualization logic, we were able to maintain Go's performance advantages while still providing meaningful visual representations.

Recommendations

To further enhance the accessibility and user-friendliness of the CPU scheduling program, we recommend implementing it as a website with Go Language as the backend and a choice of frontend framework. This approach would allow users to access the program from any device with a web browser, without the need for installation or command-line usage
.
By leveraging Go's speed and efficiency on the backend, the website can maintain the program's fast response times and accurate calculations. The frontend framework can be chosen based on the desired level of interactivity, responsiveness, and visual appeal, ensuring a seamless user experience. Additionally, incorporating more scheduling algorithms and providing more customization options for users could expand the program's capabilities and appeal to a wider audience.

Overall, the successful development of this CPU scheduling program in Go Language demonstrates the language's suitability for systems programming tasks that prioritize speed and efficiency. The project's challenges and solutions highlight the importance of striking a balance between performance and functionality when designing complex applications, and the potential for further enhancing the program's accessibility and user experience through web-based implementation.