CPU SCHEDULER

Introduction:-

In modern operating systems, a CPU scheduler is a vital component responsible for managing the execution of processes. CPU Scheduling is a process that allows one process to use the CPU while another process is delayed (in standby) due to unavailability of any resources such as I / O etc, thus making full use of the CPU. The purpose of CPU Scheduling is to make the system more efficient, faster, and fairer. Through various scheduling algorithms and policies, it aims to balance multiple factors like throughput, response time, and fairness, ultimately enhancing system performance. This project aims to implement a sophisticated CPU scheduler in C++ that demonstrates the effectiveness of different scheduling algorithms through a graphical frontend interface.

Project Description:-

This project implements a dynamic CPU scheduling system that evaluates multiple scheduling algorithms and selects the most suitable one based on predefined performance criteria, specifically the average waiting time. The scheduling algorithms included are:

- First Come First Serve (FCFS)
- Shortest Job Next (SJN)
- Priority Scheduling
- Round Robin

Understanding Scheduling Algorithms:-

The effectiveness of a CPU scheduler lies in its ability to employ various scheduling algorithms based on system requirements and workload characteristics. Below are the algorithms implemented in this project:

1. First-Come, First-Served (FCFS)

Description: This is the simplest scheduling algorithm where processes are executed in the order they arrive in the ready queue.

When to Use: Suitable for batch processing where response time is not a critical factor.

2. Shortest Job Next (SJN)

Description: This algorithm selects the process with the shortest execution time next. It can reduce the average waiting time but may lead to starvation of longer processes.

When to Use: Effective in environments where the process execution times are predictable.

3. Round Robin (RR)

Description: Each process is assigned a fixed time slice (quantum) in a cyclic order, ensuring all processes get CPU time.

When to Use: Suitable for time-sharing systems where fairness and responsiveness are crucial.

4. Priority Scheduling

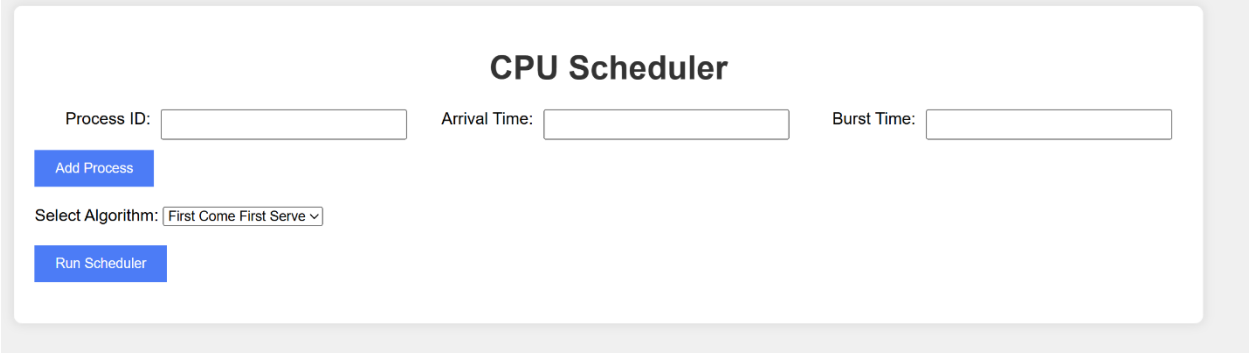
Description: Processes are executed based on their priority. Higher priority processes are selected first. This can be either preemptive or non-preemptive.

When to Use: Ideal for systems where certain tasks must be prioritized over others.

The program runs each algorithm on a set of processes, calculates their performance metrics, and dynamically chooses the algorithm that yields the best performance. This approach ensures that the most efficient scheduling method is used, enhancing overall system performance.

How to Run the Project:

- Clone the repository. (git clone website name)
- `npm install`
- Start the backend server:
    - `cd backend`
    - `node index.js`
- `npm run start`



In this you can add process and give input of their Process ID, Arrival time and Brust time , you can select the algorithm and then run the Scheduler.

Further Improvements

1. **Real-time Monitoring**: Incorporate real-time monitoring of system resources to adapt scheduling decisions dynamically.
2. **Machine Learning Integration**: Use machine learning techniques to predict process behavior and optimize scheduling decisions.
3. **Cloud Integration**: Extend the scheduler to manage processes in a distributed cloud environment, addressing scalability and resource allocation challenges.

Learning Takeaways

1. **Understanding Scheduling Algorithms**: Gained in-depth knowledge of various CPU scheduling algorithms, including their implementation and performance characteristics.
2. **Algorithm Comparison**: Learned how to evaluate and compare different algorithms based on performance metrics such as average waiting time.
3. **Dynamic Decision Making**: Developed skills in creating a system that dynamically chooses the best algorithm based on real-time performance data.
4. **C++ Programming**: Enhanced proficiency in C++ programming, including the use of data structures like vectors and queues, and standard algorithms for sorting and searching.
5. **Project Structuring**: Learned how to structure a project in C++, organizing code into multiple files and ensuring modularity and readability.

Summary

This project successfully implements a sophisticated CPU scheduler in C++, showcasing the effectiveness of various scheduling algorithms through a graphical frontend interface. The scheduler balances factors like throughput, response time, and fairness, ultimately enhancing

system performance. The graphical frontend not only visualizes the scheduling decisions but also provides valuable statistics to demonstrate the optimality of these decisions.

Resources:-

1. https://www.youtube.com/playlist?list=PLBlnK6fEyqRitWSE_AyyySWfhRgyA
2. **Low level linux kernel implementation details (very interesting) (Chapter 4)**
3. https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems/

Submitted by:- Nancy Chouksey

Enroll no.:- 22116059

Branch:- Electronics and communication engineering III yr