Elevator Operating System Scheduler README

Introduction

This README document provides an overview of the design, implementation, and evaluation of an Elevator Operating System (OS) scheduler developed for the CS 4352 course project during the Spring 2024 term. The scheduler efficiently manages elevator operations within a building to optimize people transportation, emphasizing the use of multithreading, effective communication with the Elevator OS, and an optimized scheduling algorithm.

Scheduling Algorithm

Chosen Algorithm

The implemented scheduling algorithm prioritizes assigning people to the nearest available elevator. This heuristic approach operates as follows:

1. Sort the people queue based on their start times.
2. For each person in the sorted queue:
    - Find the nearest available elevator to the person's start floor.
    - Assign the person to that elevator if space is available; otherwise, add them to the waiting list.

Justification

The chosen algorithm was selected for its simplicity and efficiency. By prioritizing the nearest elevator to each person's starting floor, the algorithm minimizes wait times and optimizes elevator utilization. Its straightforward implementation aligns well with the project's requirements for real-time responsiveness and computational efficiency.

Potential Improvements

While the chosen algorithm effectively meets the project objectives, several improvements could enhance its performance:

1. Dynamic Floor Prioritization: Consider factors such as elevator load, direction, and floor distribution to dynamically prioritize elevators for assignment.
2. Advanced Scheduling Heuristics: Implement more sophisticated scheduling strategies, such as shortest path algorithms or machine learning-based approaches, to further optimize elevator assignments.
3. Adaptive Learning: Incorporate feedback mechanisms to adapt the scheduler's behavior over time based on historical data and real-time performance metrics.
4. Concurrency Optimization: Fine-tune thread synchronization and communication mechanisms to minimize overhead and maximize parallelism, especially in high-load scenarios.

Implementation Details

The scheduler is implemented in C++ and consists of a single source code file (scheduler_os.cpp) and a makefile for compilation. The program utilizes standard C++ libraries for multithreading, file I/O, data structures, and algorithm implementation. It compiles using the GNU C++ compiler with C++11 or later support.

Conclusion

The Elevator OS scheduler developed for the CS 4352 course project demonstrates effective design, implementation, and performance within the specified requirements. By employing a heuristic scheduling algorithm that prioritizes simplicity and efficiency, the scheduler efficiently manages elevator operations and optimizes people transportation within a building. While the current solution meets project objectives, ongoing optimizations and enhancements could further improve system performance and responsiveness