Print Job Scheduling with Priority Queue Using Max-Heap

Gormery K. Wanjiru

October 29, 2024

1 Introduction

In this assignment, we explore the concept of print job scheduling using a priority queue based on a max-heap. The goal is to develop a program that allows users to add print jobs with different priorities and efficiently process them according to their priority, with higher-priority jobs processed before lower-priority ones. This system is intended to demonstrate the application of max-heap in real-world scenarios where tasks require prioritization.

2 System Design

The system is built around three main tasks:

1. **Task 1: Priority Queue Design** - Implement a priority queue using a max-heap to manage print jobs by priority.
2. **Task 2: Job Addition and Display** - Enable users to add print jobs with a user-friendly interface and automatically display the highest-priority job after each insertion.
3. **Task 3: Priority Update** - Allow users to update the priority of an existing job and reposition it within the max-heap.

3 Algorithm Details

3.1 Max-Heap Priority Queue for Print Job Scheduling

The max-heap structure allows efficient management of insertion and extraction operations based on priority. Each print job has a unique name and an integer priority, with higher integers indicating higher priority. The max-heap properties are maintained during insertion and after updating priorities.

3.2 Task Breakdown

- **Task 1** - Design the max-heap structure for print job scheduling and ensure that higher-priority jobs are processed before lower-priority jobs. Handle invalid inputs gracefully.
- **Task 2** - Implement a user interface for job insertion, displaying the job with the highest priority after each addition. Prevent duplicate job names.
- **Task 3** - Add functionality to update a job's priority, repositioning it in the max-heap based on the new priority.

4 Implementation

The code below illustrates the main components of the priority queue using a max-heap and its associated methods.

```cpp
// Priority Queue Implementation
#include <iostream>
#include <vector>
#include <string>
#include <unordered_map>

struct PrintJob {
    std::string name;
    int priority;
    PrintJob(const std::string& name, int priority) : name(name), priority(
        priority) {}
};

class PriorityQueue {
private:
    std::vector<PrintJob> heap;
    std::unordered_map<std::string, int> jobMap;

    void heapifyUp(int index);
    void heapifyDown(int index);
public:
    void insertJob(const std::string& name, int priority);
    void processJob();
    void updatePriority(const std::string& name, int newPriority);
    void displayHighestPriorityJob();
    void displayQueue();
};
```

5 Example Usage

The following input sequence demonstrates adding, displaying, and processing jobs in the queue:

```
Insert Print Job:
Job Name: "Thesis", Priority: 4
Queue: ["Thesis"] | Highest priority: Thesis (Priority: 4)

Insert Print Job:
Job Name: "Project", Priority: 5
Queue: ["Project", "Thesis"] | Highest priority: Project (Priority: 5)

Update Job Priority:
Update "Thesis" to Priority 7
Queue: ["Thesis", "Project"] | Highest priority: Thesis (Priority: 7)
```

6 Error Handling and Edge Cases

The system includes error handling for:

- Duplicate job names.
- Invalid priority values.
- Attempting to update non-existent jobs.

7 Testing and Results

Each feature was tested individually and in sequence, following the example input to ensure correctness. The results match the expected output for each task, confirming the priority queue's effective scheduling.

8 Conclusion

This project demonstrates the use of a max-heap-based priority queue for efficiently managing and scheduling tasks. By supporting dynamic priority updates and job insertion, the program provides a foundational example of priority-based task scheduling in C++.

References

- Cormen, T. H., et al. (2009). *Introduction to Algorithms*, 3rd ed. MIT Press.
- Sedgewick, R. (2011). *Algorithms in C++*. Addison-Wesley.
- Documentation: `https://en.cppreference.com/`