Oscar Lopez

CSUSM: CS 433 : Section 1

Dr. Xiaoyu Zhang

02-06-2025

Overview

This report analyzes the **Process Control Block (PCB) Table and Ready Queue System**, detailing its efficiency, optimizations, and computational complexity. The system utilizes a **priority queue (max heap) for process scheduling** and an **array-based PCB Table for process management**. The goal is to maintain an optimal balance between **speed, memory usage, and scalability**.

Data Structures and Optimizations

PCB Table (`PCBTable`)

The PCBTable stores an array of pointers to PCB objects and provides constant time (`O(1)`) access to processes using direct indexing.

Optimizations:

- Uses dynamic memory allocation (`new PCB*[]`) for flexible storage.
- Ensures proper memory deallocation in the destructor (`~PCBTable()`).
- Avoids fragmentation with a structured array-based implementation.

Ready Queue (`ReadyQueue`)

The ReadyQueue implements a priority queue using a max heap, ensuring that higher-priority processes are scheduled before lower-priority ones.

Optimizations:

- Uses heapify operations (`heapifyUp()` and `heapifyDown()`) to maintain heap order efficiently.
- Implements dynamic allocation of the heap (`heap = new PCB*[capacity]`).
- Copy constructor and assignment operator prevent memory leaks.

Logarithmic Runtime Analysis

Heap Operations (Priority Queue Complexity)

The max heap guarantees logarithmic runtime for key operations:

Operation	Time Complexity
Insertion (`addPCB()`)	**O(log n)**
Deletion (`removePCB()`)	**O(log n)**
Access highest-priority PCB (`heap[0]`)	**O(1)**

Insertion (`heapifyUp()`) moves newly added PCBs logarithmically (`O(log n)`) up the heap, while removal (`heapifyDown()`) extracts the highest-priority PCB, ensuring efficient scheduling.

PCB Table Operations Complexity

Operation	Time Complexity
Retrieve PCB (`getPCB()`)	**O(1)**
Add PCB (`addPCB()`)	**O(1)**

Array indexing enables fast lookups (`O(1)`), making PCB retrieval efficient.

Efficiency and Performance Considerations

Space Complexity

Data Structure	Space Complexity
PCBTable (`PCBTable`)	**O(n)**
ReadyQueue (Heap-based priority queue)	**O(n)**

Heap scales dynamically, efficiently utilizing memory, while the PCBTable preallocates storage (`O(n)`), reducing flexibility but ensuring fast access.

Scalability Considerations

- The heap-based ReadyQueue scales logarithmically (`O(log n)`), making it efficient for large workloads.
- The PCBTable provides instant lookups (`O(1)`) but is limited by static array sizing.

Potential Bottlenecks & Further Optimizations

1. **Heap Resizing**: The fixed-size heap limits scalability. **Optimization:** Implement dynamic resizing (`std::vector<PCB*>`) for adaptive growth.

2. **Memory Management in PCBTable**: Each `PCB` is allocated separately, leading to fragmentation. **Optimization:** Use memory pools or preallocated buffers to improve efficiency.

Conclusion

This system provides efficient process scheduling using:

- Array-based PCB storage (`O(1)` lookups).
- Heap-based priority queue (`O(log n)` insertions/removals).