Nicholas Everekyan, Kevin Gallagher
CS433: Operating Systems
Professor Zhang
September 10th, 2024

Programming Assignment 1 Report

Problem Description:

Our problem involves simulating a priority queue for managing processes within an operating system. The program implements a ReadyQueue, where processes represented by Process Control Blocks (PCBs) enter a "ready" state and are scheduled to run based on their priority. The functions include adding PCBs to the queue, removing the one with the highest priority, determining the queue size, and displaying all PCBs.

Program Design:

The design of the program involves constructing a ReadyQueue that behaves as a priority queue. We utilized the trickle-up and reheapify algorithms to maintain the heap property, ensuring that the process with the highest priority is always at the front. The challenges we came upon when developing these functions was making sure our logic contained all the possible ranges necessary to prevent accessing non existent PCBs, as well as making sure it was robust enough to fit all of the proper tests.

Choice of Data Structure:

The data structure we chose to implement for our ReadyQueue was a binary heap. With a binary heap, we can assure that processes with the highest priority can be quickly accessed, which is essential for scheduling in operating systems. Binary heaps also provide ample resources between insertion and removal operations, which are common in a ready queue scenario.

Expected Time Complexity:

Insertion (addPCB): In a binary heap, inserting an element (a new PCB) takes O(logn) time, where n is the number of processes in the queue. This is because the heap maintains its balanced structure after each insertion.

Removal (removePCB): Removing the highest-priority element, which is the root of the heap, also takes O(logn) time. This includes removing the root and then calling reheapify to maintain its properties.

Accessing the Highest Priority Element: Accessing the root element (the process with the highest priority) is O(1), as it's always at the top of the heap.

Size and Display Operations: Checking the size of the queue is O(1), and displaying all elements would take O(n).

Timing Results:

With using a binary heap, the insertion and removal operations scale logarithmically. This means that as the number of processes grows, the time for each operation will increase. There would only be noticeable delays for a very large number of processes. Here are a set of results of time from test2 without doing any optimization in the Makefile:

- Time taken: 0.040151 seconds
- Time taken: 0.0411131 seconds
- Time taken: 0.0391007 seconds

Average: 0.0401216 seconds

Due to the scalability of binary heap, the timing measures would align with an expected O(logn) behavior.

Conclusion:

Our program successfully solves the problem of managing PCBs in a priority queue using binary heap data structure. The lessons we learned from our assignment was understanding the importance of handling edge cases like array indexing and proper range checking for pointers. One thing we could improve in the long term is using dynamic arrays instead of fixed-size arrays for scalability.