Ben Foltz-Miranda

Dr. Zhang

CS 433-02

09 February 2024

# Assignment 1 Program Report

## Problem Description

The problem that I am solving is implementing a priority queue of processes in the form of the ReadyQueue class. In my ReadyQueue class I chose to use an array as the structure for the priority queue. The ReadyQueue class can add and remove process, list how many processes are in the queue and display all the processes in the queue. I also had to implement the PCBTable class which holds all the processes that would be running on the computer, when the process is ready to be run, it gets added to the ReadyQueue to be run when it is the processes’ turn based on its priority. The PCBTable class holds PCBs which are defined in the PCB class. Each PCB has a unique process ID, a priority between 1-50, and a state which describes what the PCB is currently doing, the possible states are new, ready, running, waiting, and terminated. With these three classes, we can demonstrate the way that operating systems schedule processes.

## Program Design

For the PCBTable class, I used a vector of PCB pointers to hold all the processes. The PCBTable class didn’t seem like it needed anything complicated to hold the processes that were going to be added to it so I thought that it would just make more sense to keep it simple. I did use a vector instead of an array because this allows the size of the list of processes to increase without having to create a new array with a larger size and move all the processes to the new array.

For the ReadyQueue class, I used an array as the base for the priority queue. I chose an array as the base for my implementation of the priority queue because it seemed like the simplest data structure to be able to implement a priority queue without using a STL class. It also seemed like it would make the most sense to use an array because since it is a priority queue, we would always remove either the front or the end of the queue depending on which we mark as the highest priority. The ReadyQueue class has two data members, an array of pointer PCBs called queue, and an int called count, which holds how many PCBs are in the ReadyQueue. The time complexity to add an element is O(n), when adding an element with the highest priority it won’t have to make n comparisons, but it will have to shift the elements in the array to the right since the front of the array holds the PCBs with the highest priority. When adding an element with the lowest priority, it can add it to the queue in O(1) since it just has to check the priority of the last element in the array and then if it is lower than the process gets added to the end of the array. When adding an element between the highest priority process and the lowest priority process, it will have to loop through the array until the PCB being added is greater than the PCB that is currently being looked at, then it must shift all the elements to the right and then insert the new PCB into the ReadyQueue. For every PCB that is added to the priority queue, it’s state is changed to READY. When removing a process from the priority queue the first element is copied into a pointer PCB and then all the elements are shifted one to the left, when the end of the priority queue is reached, the last element is set to a null pointer since we don’t need two copies of the last element. After the PCBs have been shifted to the left, the count is decremented since a PCB is being removed then the pointer to the highest priority PCB’s state is changed to RUNNING and the PCB is returned.

## System Implementation

I did run into some problems during my implementation of the ReadyQueue, I was having trouble figuring out where I had a memory leak. I was able to figuring it out by reading through the error given by the Valgrind memory test on GradeScope. The issue was that when I was removing a PCB from the ReadyQueue I had forgotten to set the last PCB in the queue to be a null pointer after I had shifted all the elements towards the front of the array.

## Results

I believe that all the features that were required are included in my submission. I think that something that I would like to improve on would be implementing a different data structure that would make it more efficient to add and remove element from the ReadyQueue because the way that I implemented it works well for a small amount of processes, but assuming that could be an extreme amount of processes running or being added to the ReadyQueue, it could really slow down. The ReadyQueue was able to run under 0.1 seconds on the second test.

## Conclusion

I was able to solve the intend problem successfully. The program is able to successfully create PCBs and manage them with the PCBTable class and the ReadyQueue class. A lesson that I’ve learned from this assignment is that I should be more careful when it comes to using pointers because it can be very easy to have a memory leak. I liked that this project had a lot of the structure of the classes filled out and we just had to fill in the parts that required thinking. I think the only thing that I had a bit of trouble with was that it was a bit confusing on how all the classes went together to make the whole program, I think it could have just been something that was my fault for not fully comprehending the description of the assignment, but in the end I was able to figure it out and get it to a point that I am satisfied with.