Elaeth Lilagan, Kyrstn Hall

Dr. Xiaoyu Zhang

CS 433 – Operating Systems

4 September 2022

Assignment 1 Report

Submitted Files:

- **pcb.h**
- **pcbtable.h**
- **pcbtable.cpp**
- **readyqueue.h**
- **readyqueue.cpp**

How to Compile/Run the Program:

To compile, command used: make

To run, command used:

- ./test1 && ./test2
- g++ test1.cpp (or test2.cpp) pcbtable.cpp readyqueue.cpp ⇒ followed by ./a.out
    - Unfortunately, this method was time consuming, so then the command to use most of the time would be ./test1 && ./test2

Results and runtime of test 2:

Our program ran both test 1 and 2 with all the supported files in complete fashion. Nevertheless, we ran the program 5 times to get an exact average time to calculate the following table below.

Time 1	**Time 2**	**Time 3**	**Time 4**	**Time 5**
0.0614686 seconds	0.0993106 seconds	0.0502064 seconds	0.0572126 seconds	0.104009 seconds

With this in mind, the average execution time is expected to be 0.07444144 seconds.

Features Implemented (via. class):

1. PCB - includes an enum class to represent the following process states in the system, which includes an ID, priority, and state.
    a. Constructor sets a this pointer of the following variables in the public PCB class.
    b. Each variable has its own get() to return the 'this' pointer of the variable.
    c. Both priority and state will have a set() that will have a 'this' pointer assigned to the variable.
    d. display() will output the following process states that corresponds to the following cases (will be used when its on the readyqueue.cpp)
2. PCBTable - This will use a vector array of all PCB elements inside of the system, which have the variables, size (checks how many PCBs are in the table) and processes (vector array.)
    a. Constructor includes the vector array that has the resize function to check the sizes available inside the table (array pointer.)
    b. Destructor includes a for loop to go through the size of the vector array, which will delete the elements inside the array and will set the array to NULL once all the elements have been deleted. Clear the array after the loop and set the size to 0.
    c. getPCB() will return the following element in a given index.
    d. addPCB() will use the insert function of the vector stl class which will iterate the fronts and ends at index PCB, then iterate the size.
3. ReadyQueue - Queue of PCB's that are in the READY state to be scheduled to run, which will be a priority queue of the process with the highest priority to be selected next. Variables consist of the PCB* array and count.
    a. Constructor must include ': Q{}' on the function name to declare the empty array to be used in the file and initialize the count to default (or zero.)
    b. addPCB() will add a PCB to the queue by first checking the queue being full. If not, then set the Q array to the passed pointer parameter, increment the count, and set the state to ready to trickleup() the elements within the PCB.
    c. removePCB() will remove a PCB from the queue by first checking the count so it can return NULL. If there are elements inside the queue, we set a variable to the root to return the removed element, set the root assigned to the last job, decrement the count, and set the state to running. We then set two more variables to keep track of the current index and to hold the location of the bigger child. After initialization, we set a while loop that runs to see if the current index exceeds the count by getting the bigger child of the parent. Next we set an if statement that checks if the child is not -1 and the bigger child is greater than the current index, which will swap both elements and update the current index by assigning it to the bigger child, and set an else statement for no need of swapping. After the while loop, we would then return the element that has been removed from the PCB.
    d. size() will return the length of the array.

e. displayAll() will set the following respective statements when the queue is empty or not.
f. getBiggerChild() will return the following child by highest priority.
g. trickleup() used for having an element go up the heap by highest priority (swap with high value with root.)

Additional:

- Global variable ‘count’ allows an adjustable size for the ReadyQueue
- Vector PCB* processes used to store PCBs to include within the system for pcbtable.h

Choice of Data Structure: Initially, we planned to use vector as our data structure since there will be a set of values to be stored in the following indexes by deleting and adding, however, we were restricted to not use the vector standard library, so we decided to use heap. EJ also considered using a hash table because we would use arrays and thought of an idea for getting the value of that priority and swap, but then it would require lots of space complexity, which will affect the time complexity and performance of the program. Kyrstn had an idea that if we were to use max heap, the time complexity would be much quicker, and if we were to use a tree (BST), it would be much slower to keep up with the time. Heap, in addition, would be the best way to implement a priority queue.

Time Complexity of Implementation:

Insertion (best, worst): O(1) , O(Log N)

Deletion (best, worst): O(1) , O(Log N)

Lessons Learned/Re-Learned:

- Elaeth Lilagan
    - Not having practice with data structures and using different programming languages (such as Java, Python, etc.) for the past year was a huge wake up call to catch up on. At first looking through the guidelines was a bit overwhelming, but then having my partner refresh the things that we will use to implement this program benefited me to get the whole program working. Familiarizing myself again with pointers and constructors was the benefit of understanding how to call certain variables from public/private and determining what algorithm to use. With multiple attempts to run and debug, it was a great experience to review the benefits and drawbacks of certain data structures.
- Kyrstn Hall
    - At first, me and my partner were overwhelmed by the instructions and were unsure where to start. Eventually, we started to understand how we wanted to

implement our program and what kind of data structures we would like to use. We even utilized office hours to see if we were headed towards the right direction and asked for clarification on some topics which were beneficial. Personally, I thought the hardest part of this assignment was debugging. It was difficult to trace our errors at first, especially the segmentation faults, but after testing our code with gradescope, we had an easier time understanding where the errors were occurring and why. In the end, it felt satisfying to be able to trace the bugs and fix them. It was also nice to be able to bounce off ideas with a partner on how to implement a certain function, or ideas on how to fix an error.

References/Resources:

In our readyqueue.cpp, we were able to reference some of the code we had written in CS 311 to implement Heap. Aside from that, we utilized old CS 311 notes and many online sources to help solidify our understanding of topics such as max heap, vectors and pointers, and much more. Some links that we read through include:

- How the PCB works
- Vector library: Vector Functions and Vector have dynamic arrays
- PCB can be simulated using Heaps however I will be using a different format
- https://stackoverflow.com/questions/7397768/choice-between-vectorresize-and-vectorreserve
- How to initialize a pointer array in constructor