Salma Rashed and Taya Wongsalong

CS433: Operating Systems

Dr. Xiaoyu Zhang

September 19, 2022

Assignment 1 Report

Submitted Files:

- **PCB.h** : A header file that declares the class for PCB and PCB Table objects
- **PCB.cpp**: A source file that contains the functions for the PCB and PCB Table classes
- **ReadyQueue.h:** A header file that declares the class for ReadyQueue objects
- **ReadyQueue.cpp:** A source file that contains the functions for the ReadyQueue class
- **Main.cpp:** A source file that contains the code for test 1 and test 2
- **Makefile:** Allows us to compile the above files using the make command and creates executable prog1, also creates executables (.o) for all of the source files

How to Compile and Run the Program:

- To compile the program, use command: make(in shell) or g++ test1.cpp readyqueue.cpp pcbtable.cpp (for test1), and g++ test2.cpp readyqueue.cpp pcbtable.cpp(for test2).
- To run the program, use command: ./test1 and ./test2 (with make on shell) or ./a.out

Results and runtime of test 2:

Our program was able to successfully complete test 1 and 2 and all of our classes are functioning well to the best of our knowledge. To get the average runtime of the loop of 20,000 iterations in test 2, we ran the program 5 times and took the average.

Time 1	Time 2	Time 3	Time 4	Time 5
0.0797407 seconds	0.0874004 seconds	0.0441989 seconds	0.0728523 seconds	0.116543 seconds

Using this data we determined that the average time is 0.0797407 +0.0874004 +0.0441989 +0.0728523 +0.116543 =0.4007353/5= **0.08014706 seconds**

References:

- We used CS311 notes to review the heap array implementation . We also used previously written code from HW5 that we coded in CS 311 with some updates , we specifically used trickle up and getParent fu in the add function and updated them to perform better in out program
- We also used this **source** to make sure we had the right pseudo code before coding ReadyQueue.cpp.
- We also had an error in the cpp checktest so we used this **source** too so we could fix the error in the constructor .
- We had a minor problem in our destructor so we used theses two sources: **source1** and **source 2** to figure out how to implement the auto ptr**.**

Features Implemented

- **PCB.cpp**: The first object type that we had to define was PCB .Each PCB consists of the following attributes: ID number, the last declared PCB's IDnumber (last_id), priority number, state, and a bool inQueue that tells us whether the PCB is in a ReadyQueue or not.
- **PCBTable.cpp :** Our PCBTable class is very simple and consists of an array (named processes_in) Of 20 PCBs.

*PSBTable.cpp had a getPCB function that helped us get the PCB at index "idx" of the PCBTable. We did that by just returning the pointer to the PCB at index "idx"

*Last thing we had in PCBTable.cpp was the add function that had a purpose of adding a PCB in the PCBTable we setted the index in processes_in to pcb.

- **ReadyQueue.cpp and ReadyQueue.h:** In this function, we have a functions that we've implemented from ReadyQueue.h. . Those functions were declared in ReadyQueue.h include *addPCB(PCB* pcbPtr)*, *removePCB(), size(), trickleUp(), swap(l1, l2), getSmallerChild(), reheapify(), getParent(), and even()*. We also have a constructor, ReadyQueue(), a copy constructor, ReadyQueue(), and a destructor, ~ReadyQueue(); they are all defined in the public class.For our private class, we declared a constan,t MAX_CONST = 2000, declared a heap with an array Implementation, PCB* heap[] which has all of the READY processes as the state, and lastly we also declared the current size of the heap called *int curr_size*.

Breakdown implementation explanation of ReadyQueue.cpp:

For our addPCB(), we initialize the heap array with curr_size as the parameter with the pcbPtr, then set the state to READY and used the additional function trickleUp() which will "trickle" up to swap with The parent/root. Then we increment the curr_size by 1 so the positions get swapped.

In the even() function, we want to get an even number so our logic was that we take the mod of 2 and Check to see if that is equal to 0, if so, then return true. Otherwise, return false.

In the getParent(), we want to find the location of the parent, and we will also call the even() that was implemented earlier. We declared int parent, then keep iterating if the child location is even, if not it is odd and the parent's location will be returned.

For trickeUp(), we started initializing x, which is the last job's location from the heap, and decrement x's Last position. Then we used a while loop to keep loooping as long as x is greater than 0. If the position of The heap's current priority is larger than the parent's position priority then we call the swap function to swap The current position with the parent's location based on the child's location. Then, the loop will stop and while The last job's location is NOT greater than 0, the loop will exit out of the function.

For getSmallerChild( ),we started by locating locations for children : LC and RC then we did if,else if and else statements that can take care of all the case such as returning the location of the smaller child, taking care if both beyond curr_size-1 by returning as special value or even if just the RC is beyond curr_size-1.We also dealt with the regular case where both aren't beyond curr_size-1 which returns the RC in the end

For reheapify(), we intialize a value X that will hold our current location , then we move the last job to the front and decrement our current size . We later intialized X to 0 which makes as at the root . Then we Do a while loope to keep on looping as long as X is within the array. Then we initialize the child so we could find the location of the smaller child by calling the getSmallerchild function that we worked on earlier Then we proceed by having an if statement that will check if the location of both children are off the tree, stop the loop and if the smaller child is larger than the parent value,then it will swap child with X and set X to child , if there are no swaps it stops the loops and returns

For removePCB(): we started by changing the state to RUNNING when removing the PCB from the queue.We then created a PCB temporary pointer to point at heap[0] which is the last job then we called the reheapify function and returned the temp value swap(int l1, int l2): our Swap function’s purpose is that we are trying to swap the positions of two PCB's on the PCB array. We started by declaring a PCB pointer and setting it to the first pcb location, then we set the The first pcb location to the second pcb position , and finally setting that second location to the pointer we declared earlier .

For size(): all we did is return the number of elements in the queue

For displayAll(): we did that by doing a foor loop and calling the display function that was already implemented for us in pcb.h file

For isEmpty():checks if the queue is empty then returns if it is.We did that by returning the current size as 0