Programming Assignment 1 Report

Submitted Files:

- **pcb.h**: A header file that declares the class for pcb and pcbtable objects
- **pcbtable.h**: A header file that declares the class for pcbtable objects
- **pcbtable.cpp**: A source file that contains the functions for the pcb and pcbtable 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
- **Makefile**: Allows for automation for having to compile and link object files of the above files using the "make" command and creates executable files test1 and test2, also creates object files (.o) for all the source files

How to compile and Run the Program:

To compile the program, use command: make

To run the program, use command: ./test1; ./test2

Results and runtime of tests 1 and 2:

Our program successfully completed tests 1 and 2 and all our classes are functioning to the extent of what we know. To get the average runtime in test 2, we ran the program 5 times and took the average.

Time 1	**Time 2**	**Time 3**	**Time 4**	**Time 5**
0.0804017	0.0867147	0.0867283	0.0865214	0.0886946

Utilizing this data, we determined the average execution time to be 0.08581214 seconds. These were the average times we expected, as all the times listed were less than 0.1 and around 0.08 seconds.

Time Complexity:

The time complexity of the readyqueue's addPCB implementation is O(1) when added to the end of the vector. Similarly is removePCB, which is also O(1). HeapifyUp implementation is O(n log n), and heapifyDown implementation is O(n). Both provide the sift up and sift down operations of the heap.

Features Implemented (by class):

- PCB– We defined the PCB object type. Each PCB contains the following attributes: an ID number, priority value and a process state.

PCB contains a constructor that initializes each of the PCB attributes of its invoking PCB object.

- PCBTable consists of a private instance variable, a vector of PCB pointers labeled 'table'.

PCBTable consists of a constructor and destructor, where the constructor initializes the invoking object's values through its parameters. The destructor destroys each PCB object and clears the vector to contain a size of 0.

- ReadyQueue – Our ReadyQueue class is a priority queue implemented as a binary heap. We used an array of PCB pointers labeled as 'queue'. We utilized this so that the ReadyQueue can access and adjust contents of each PCB in the PCBTable by reference.

The ReadyQueue contains attributes: PCB* Q[MAX] {}, this provides us the PCBs in the queue.

The ReadyQueue helps provide and maintain the structure of the heap and works with some of the predefined and already given functions in the assignment. The ReadyQueue uses a default constructor that initializes the invoking instance variable labeled 'size'. We also provided definitions from the given member methods signatures from the assignment, like displayAll(), heapifyUp(), heapifyDown(), etc...

Additional Features:

- Global variable MAX to allow an adjustable size for the ReadyQueue

Design and Implementation choices:

Once the project was assigned, we immediately decided to implement the ReadyQueue and applied the binary heap. We thought that the PCBTable would be very efficient when we were to assign processes utilizing the binary heap. We did what the assignment mentioned. Near the end of wrapping up the assignment, we implemented the PCB table destructor to deallocate the heap memory of each PCB in the instance variable of PCBTable, labeled 'table'.

We knew that if we approached this assignment with a linked list, it would be more time-consuming. As over time, working with creating the binary heap attributed our understanding of the tasks that we were needing to accomplish. So, the simple static array was the best approach as it would be less time-consuming and easier to understand to approach the ReadyQueue.

Overall understanding and learning outcomes:

From this assignment, we learned how to manage our time better, as we started once the assignment was made available to us. We managed to fix the observable bugs. Some of the design flaws we had were related to our approach mixed with our understanding of the assignment. However, over time, we began to gradually have a full understanding. We were able to re-learn many concepts relating to data structures and algorithms, and we learned that its usage is incredibly fundamental towards the project. We also learned about operating system concepts relating to the understanding and implementation of the PCB (Process Control Block) and various terminologies and implementations that are added to its operations.

References:

We used previously written code from CS311 to refamiliarize ourselves with data structures, and implement the concepts of what we currently recognize for both operating systems and data structures into C++.

Future improvements:

If time is given to us, we can improve and implement more features and provide more optimizations. We noticed that the implementation of using the array and allocating a large sum of space into the array may be inefficient. We may also better organize our code and maybe start thinking of providing a coding style when or if we do delve into investing a lot of our time into this project.