Programming Assignment 1 Report

Andres Hinojosa

Submitted Files:

- **pcb.h:** The header file containing the PCB class
- **pcbtable.h**: The header file containing a table of PCB objects using a vector
- **pcbtable.cpp**: The implementation file of the PCB table functions
- **readyqueue.h**: The header for the ready queue, using a binary tree
- **readyqueue**.cpp: The implementation file for ready queue which defines the functions

How to compile and run the program:

- To compile the program, use command: make
- To run the program, use command: ./test1 or ./test2

Results and runtime for test2:

- My program was able to complete test 1 and test2 correctly. I am confident that all my member function for PCB and PCB table work correctly, but my implementation of the binary tree in ready queue could be better. The following table shows run times for test2:

Time 1	Time 2	Time 3	Time 4	Time 5
0.0752812 seconds	0.0745715 seconds	0.0735079 seconds	0.0754145 seconds	0.0745605 seconds

- According to the data, the average time the program took was 0.07455712 seconds.

Features implemented:

- **PCB**: I started off with the PCB class since it is the simplest and all other files use PCB objects. The data members include an id, a priority, and a state. All the functions either change these attributes or return them. The display function displays all its attributes at once.
- **PCBTable**: The next files I worked on were the pcbtable header and implementation files. I decided to go for a vector array since I believe it makes the most sense and does not require complicated functions or sorting. I created the vector as a private member of the class and using the constructor, I was able to reserve the amount of space required for the size. For the destructor, I simply made a loop that goes through and deletes all objects until it is empty. I used the operator overload of [] to return a PCB pointer or add one for the other functions.
- **ReadyQueue**: ReadyQueue was the final header and implementation files I worked on as it looked and was the most challenging. I understood that this would be used on test2, which would require an efficient execution time. Because of this, I decided to go for a binary tree of pointers as my data structure. I added a vertex node with pointers to their children and parent, as well as a spot for the PCB pointer. When adding PCB's, they are sorted by their priority number. Each time an add or remove function is called, a counter declared in the class is incremented or decremented to save time instead of manually counting each vertex.

Design and implementation choices:

When I worked on the PCB Table, I first thought about using a binary tree since it would be quick to go through, but after carefully looking at the functions, I decided that it made more sense to make a vector as it was recommended by the comment on the program and due to the overloaded [] operator. The data structure for the PCB Table sounds like it should not be complicated for its purpose. Next, when I started the Ready Queue, I also began by thinking it would be a good idea to implement a binary tree using pointers as it gives a quick search time. Because of this, I went ahead and implemented it and I was able to get times of under 0.1 seconds.

Lessons Learned:

The most obvious thing that I have learned of this assignment is to start early. I thought because we had a lot of time, I would be able to do it later. With more time however, I would have much more time to rethink decisions I have made to better my program and come up with the fastest solutions. Another thing I could have done was to freshen up on data structures and algorithms since only a few data structures came to my mind when doing the assignment. Finally, an important thing I should have done was to communicate with others for more perspectives and to learn.

References:

A reference that I used were documentations and stack overflow for vectors, as I had an idea of what kind of functions I would use, but I was not sure what their names were and to look for the ones that return the items I was looking for. For implementing the binary tree, I looked at one of my program assignments that I created in CS 311 and used it to implement PCB pointers.

Future improvements:

To improve my program, I would add a BST function to my binary tree. A tree has a fast time, but it will not always if it is leaning towards one side or another. Having a tree that balances itself out would give an extremely fast time complexity of logn. Other than that, I would make my code cleaner and give some more time for optimizing in smaller ways.