CS475: Lab 7 Code Review

# Questions to answer:

1. Who did you work with? Caleb Eby, Nozomu Ohno
2. How is time defined?   
   Time is incremented each cycle (the clock tick). Each processor does one chunk of work, any number of io processes could happen at the same time. (more on that in the next question)
3. What kinds of things can happen at each “moment”?  
   In each clock tick, look at everything in io queue, increment each one's burst time so one less IO task to do. Any IO processes that finished with IO go to ready queue. if IO is not scheduled yet put in ready queue (it is a cpu burst), scheduler algorithm operates on the queues that are set up with this.
4. What happens when the parameters of the simulation changes? Do the results make sense?  
   Round robin time quantum and number of processors were changed. There are 1, 2 and 4 processors. The time quantum was 1, 2,3,5,10,20 milliseconds. FCFS near infinite time quantum since it operates as long as the burst time is. Wait times and turnaround times match what is expected with differing number of processors. Response time increased as well, being lower with more processors and higher with less processors..
5. Were there any bugs or issues that you fixed?  
   c++ memory issues, issues with processors printing strange, unexpected values. process would be done but processor would keep running processes that were done. Implemented checks of if the cpu was ready and if the io was ready, and if a process completes it is "done". If a process is done, leave the system. Additionally, there were issues with code being written in the wrong order, and with forgetting to push or pull from queues. Basic debugging solved most issues (printing values and checking if they match).
6. Explain how each data structure is used.  
   map for process table, queue for ready queue, set for io queue, sets for deleting items from io queue. Sets for things that don't really have an order (io processes). vectors for execution history of each processor, to help with the bug of processes running if they are done.
7. What alternative data structures were considered? Why were they not chosen?  
   IO was intially handled as a queue, but because of the limitations of queues (being unable to access elements outside of the first), decided to make it a set, since order did not matter and sorting would be used. They also used ENUM type to help with process states (such as NEW READY WAITING RUNNING and EXIT).
8. What alternative data structures would you have also considered?  
   All data structures were considered, but based on pros and cons of each data structure and implementation choices, the data structures from question 5 were the final choices.
9. Is memory managed correctly (e.g. are pointers correctly deallocated)?  
   Not really, but it does not matter for this use case because as processes are done their pointers are deallocated.
10. Are exceptions handled correctly?  
    One manual exception of when the processor executes a process after it is done, since that is not supposed to happen.
11. What took the longest in general (our own question)?: Round robin, it was a lot harder to implement this algorithm than expected, and the set time quantum complicated time calculations.