The biggest problem I faced when developing my project was trying to figure out the order that each insertion, removal , and incrementation operation should be done. For example, should the overall time be incremented before a process that needs to do IO is put onto the IO queue, should a process that is finished IO and needs to get back onto the ready queue do so before a process that finished executing it's CPU burst but needs to execute another one. I overcame this problem by constantly debugging my program to see which order of operations causes the program to perform the way I expect it to perform. I did this by creating a debug function which outputs the state of the currently running process during every time cycle.

The second biggest problem I faced was figuring out how to do context switches correctly. I had to figure out a way to have the program I developed wait a specified amount of time from the time that the currently running process has either finished, went to do IO, or has been preempted, to the time that the next process in the ready queue becomes the currently running process. I solved this problem by giving the process class a BLOCK state flag which , when set by the currently running process, tells the context switch function to decrement the contextSwitchDelayTimer variable and only assign the next Process from the ready queue to the currently running process when contextSwitchDelayTimer is equal to zero.

The third biggest problem I encountered was that in most programs that I write I use ints to store numerical values. In this program, I realized that my figures weren't correct and spent many hours trying to find out the cause of the problem. I discovered that my values were so large that they exceeded the bounds of what an int can hold. For this reason, all the variables that hold very large numerical values (such as total turnaround time) are stored used unsigned long longs, since that is the largest primitive data type that exists for c++.

The fourth biggest problem I encountered was random bugs that I encountered whenever I changed something in my program. Whenever I added something or made a change to improve performance, a new bug would creep into my program. I spent a good deal of time messing with gdb to

fix bugs that crept into my program from changes I made.

I believe that the best scheduling algorithm is shortest job first with exponential averaging because it has an average waiting time that is close to shortest job first and shortest remaining job first. Since all processes that have just executed will likely have a larger exponential average than processes waiting on the queue, the chance of starvation is very unlikely. Also, as opposed to shortest job first and shortest remaining job first, this algorithm seems to be actually implementable in reality. The limitation on the algorithm, I would assume, is that you can't assume that a trend that an algorithm follows will continue.

Increasing the size of the ready queue increases the average waiting time, decreases the turnaround time for shortest job first, shortest remaining job first, and shortest remaining job first with exponential averaging. Ready queue size doesn't affect the turnaround time and throughput for first come first serve and round robin.

With First Come First Serve Unbounded , decreasing the context switch time to 1 decreases the waiting time, increases the throughput per 1000 time units, decreases the turnaround time, and increases the CPU utilization. 99 % of the time is spent waiting. When the context switch time is increased to 10, the waiting time and turnaround time are increased. CPU Utilization is decreased. The turnaround time is decreased. 99 % of the time is spent waiting.

With Shortest Job First Unbounded, decreasing the context switch time to 1 increases CPU utilization, decreases the total waiting time, increases the throughput per 1000 time units and CPU utilization. 98 % of the turnaround time is spent waiting. Increasing the time to 10 decreases CPU utilization, increases waiting time, decreases throughput, and turnaround time is increased. 99 % of the time is spent waiting.