Workload:

Denny Shen

All graphs were generated using the .py file provided with little changes.

Graph1:

In test1, the child process is a CPU intensive job calling a function that calculates pi, and the parent process is a I/O intensive job calling large amount of printf() functions. At the beginning of the test, both processes enter the highest priority queue, q0, and, after 1 tick, they are both downgraded to the second priority queue, q1. As they consume the 2 ticks allocated for q1, they again are downgraded to the lowest priority queue, q2.

Graph2:

In test2, the fork() function is called 4 times, and there should be 16 CPU intensive processes. At the beginning of the test, all processes enter the highest priority queue, q0, and, after they consume their 1 tick allocated, they are downgraded to the second priority queue, q1. After they consume their 2 ticks allocated in q1, they are downgraded to the lowest priority queue, q2. Also, the boost functionality is shown by some of the green processes where they went from q0 to q1 to q2 between tick range 220 ~ 280, and then were boosted directly to q0 at around tick 330.

Graph3:

In test3, sleep() function is used and there is only one process. In this test, sleep() was called multiple times in a for loop so that the process always yields CPU control before time tick increases. Consequently, the process will always stay in the highest priority queue or the second highest, but will not drop down to the lowest priority queue. Duration 0s were increased to 0.1s so that it appears visible on the graph for observation, and the code were commented in the other tests.