To test the four scheduling algorithms in our program, we tried to design test cases that would provide as much code coverage as possible by exercising a wide variety of possible scheduling scenarios. In particular, our test cases were designed to induce preemption (for the preemptive scheduling algorithms), processes arriving while others are in progress (both that would and wouldn’t cause preemption at the next scheduling opportunity), and idle time. We believe we covered most of the scenarios possible; throughout testing, we frequently uncovered bugs that required careful analysis in a debugger, and sometimes significant re-working of the relevant code. In particular, one of our tougher test cases forced a last-minute re-write of the round-robin scheduling algorithm, for which we needed to fully simulate a ready queue and the behavior of a live scheduler in order to accurately reproduce its behavior.

Interestingly, for our toughest test case that we developed expressly for the round-robin algorithm, that algorithm performed the worst; non-preemptive priority scheduling produced the lowest turnaround and wait times for this test case. In fact, for most of the test cases we tried, non-preemptive priority scheduling did consistently better than the others. The two preemptive schemes didn’t, in general, do as well as the non-preemptive schemes in minimizing turnaround and wait times for our test cases. Whether this is simply an artifact of the types of test cases we chose to maximize code coverage, or a more general observation on the efficiencies of these scheduling algorithms, is not immediately clear.

Regarding ease of implementation, FCFS was the easiest to code, followed by non-preemptive and preemptive priority respectively. As indicated above, the round-robin algorithm proved the most complex to implement, having a lot of subtle nuances that eliminated most of the strategies we tried, despite intuitively appearing to be one of the most straightforward of the algorithms.