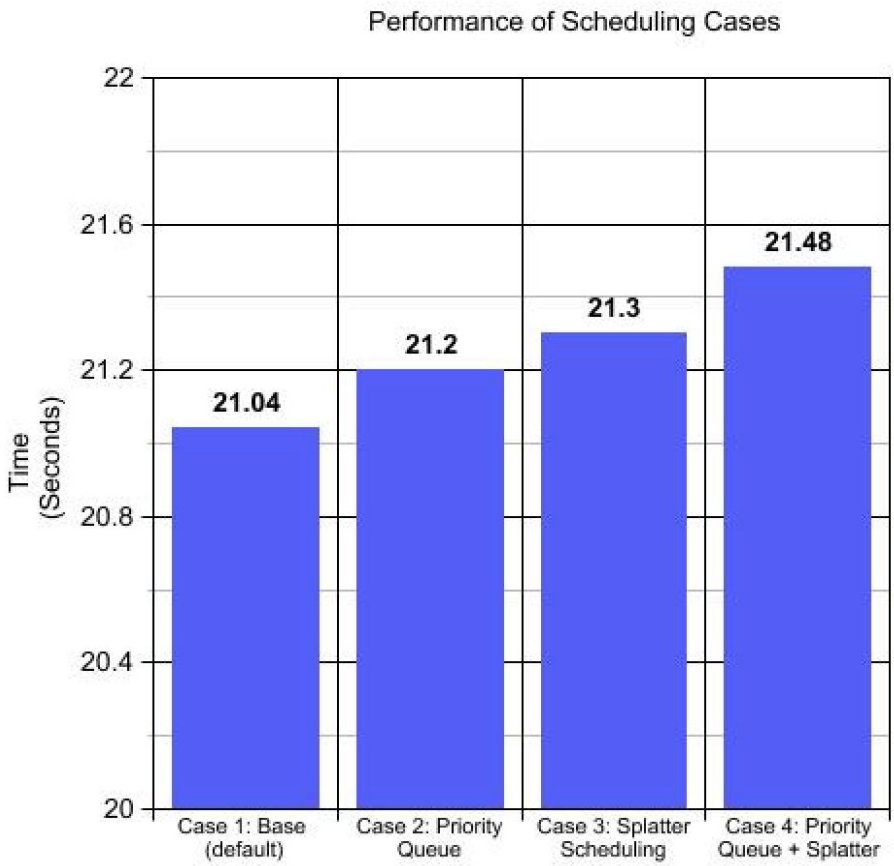
Benchmark

	Case 1: Base (default)	Case 2: Priority Queue	Case 3: Splatter Scheduling	Case 4: Priority Queue + Splatter
Run #1	21.04	20.99	21.09	21.43
Run #2	21.04	21.15	21.01	21.44
Run #3	20.97	21.55	20.97	21.40
Run #4	21.12	21.09	22.15	21.66
Average	**21.04**	**21.20**	**21.30**	**21.48**



Analysis

Here are our results from running the benchmark for each case where the numbers are in second(s). We did four runs for each case and took the averages for them. When doing the benchmarks for each scheduling case, we performed all of them in user mode and not root. The times were retrieved from using the ***time*** command with our benchmark.

Case 1:

For the default case, it should have the fastest time because the FreeBSD scheduling is already optimized by the creators, so it should already be the fastest and whatever we implemented should not be faster than this.

Case 2:

For the second case, which is just the Priority Queue, the results show that it is slower, but not by much at all, roughly around 0.16 seconds. We can see that this is true because the default scheduling algorithm runs the threads in Round Robin fashion, compared to a Priority Queue which at the end of the quantum will insert the thread back at the front of the queue rather than the end of the queue. Thus, this may cause slower performance since a thread that could have finished earlier now has to wait for the higher priority thread to finish, even if it runs past the quantum.

Case 3:

For the third case, which is just the Splatter Scheduling, the results also show that it is slower than the default case by 0.26 seconds, and the second case by 0.10 seconds. We can see that this is slower than the two cases because now the threads are assigned a random run queue instead of by it's priority, but only user processes. By choosing a random runq for a thread, the runq is random, but the scheduler always chooses the highest priority non-empty runq and selects a thread in it to run. Thus, if we choose a random runq, the thread can be placed at a much lower priority runq which means it will have to wait longer until it is selected to run.

Case 4:

The fourth case, which is Splatter Scheduling with Priority Queue, is the slowest of all of them. We have already shown that Splatter Scheduling is slower than both the Priority Queue and the Base Case. We have also shown that the Priority Queue is slower than Base Case as well. Thus, by combining both Splatter and Priority Queue, it makes sense that Case 4 will be slower than Cases 1, 2 and 3.