Project 4: Rate Monotonic Scheduler

Ariel Gutierrez

For the Rate Monotonic Scheduler project, I used a total of 5 child threads and 4 semaphores for synchronization. I used a docker container running Debian GNU/Linux as the operating system and C++ as the programming language with pthread and semaphore header files.

For synchronization, I used 5 semaphores so that each child thread had one. In the main function, I initialized the semaphores, and I initialized and created the scheduler thread. At the end of the main function, I destroyed the attribute for the scheduler thread and the semaphores, and I then printed the output to the command line. The scheduler thread creates and initializes the threads with attributes that include priority and processor affinity. All threads, including the scheduler thread, have priority as an attribute and all run on core 0. Then, the scheduler thread had a for loop that loops for 160 times and its semaphore at the beginning of the for loop. Inside the for loop is where the scheduling happens depending on which time unit it is in. When it was time for a thread to run, it would be scheduled with sem_post(), its dedicated counter would increase, and if it did not finish the counter for overruns would increase and it would be canceled. Furthermore, the threads T1, T2, T3, and T4 all had a sem_wait() call on their respective semaphores for synchronization. At the end of the for loop in the scheduler thread, a timer function is called that waits for 10 milliseconds and then sem_post() is called for the scheduler. Also, each thread (T1,T2,T3,T4) has its own mutex lock that covers variables that are shared between the scheduler and the subthreads.

I could not get the failed case to work. When I let T2 do the doWork function more than 200,000 times, the count for the number of overruns of T2 would be stuck at 79. Moreover, T3

and T4 would have no overruns. Another error I would get was that T1 would sometimes overrun even though it should not.