Allison Thompson

CPSC 380

Hansen

5/6/19

Project 3

I created a Scheduler class to define the rate monotonic scheduler. The priority was decided by the shortest cycle duration of a task to be highest priority.

The timer thread thrT is initiated in Scheduler::begin() as the first thread called. Within Scheduler::timer() I set processor affinity at the start of the function to 1. The time iterates for a cycle, and at time == 0 the timer will schedule task threads thr1, thr2, thr3, and thr4 to begin, followed by a semaphore sem1->signal() to signal the task with the lowest priority to begin. Each thread uses a semaphore to signal the next priority thread to begin. I use chrono::milliseconds to set each time period to 10 milliseconds.

I set processor affinity again in Scheduler::execute() to 1, which is a function called by each thread. Through execute(), each task thread calls the function Scheduler::doWork(). By default doWork() is called an assigned number of times based on the unit. Forced overruns will call doWork() until the overrun condition is induced. When thr4 is in overrun, the program will interrupt thr4 in order to schedule the next cycle. When thr1, thr2, or thr3 are in overrun, the program will pick up the unfinished thread in the next cycle.

When any task threads thr1, thr2, or thr3 have an overrun, the program waits until the task finishes before scheduling the next task. During Case 2 when task 2 is in forced overrun, the task did trigger task 3 and task 4 to overrun 3 times each. During Case 3 when task 3 is in forced overrun, the delay caused task 4 to go into overrun 6 times. Each time an overrun occurred, it prevented the same thread of the next cycle from triggering so it could finish the previous run. Each case is outputted as Case\_.out where \_ is equals to case number.

